Classification of remotely sensed imagery for assessing machine learning algorithms

Lars Doorenbos

Supervisors:
Dr. Erzsébet Merényi
Dr. Michael Biehl
Dr. Kerstin Bunte

August 2020

Abstract

The information present in remotely sensed hyperspectral imagery allows for the subtle discrimination between land covers, with applications in domains such as agriculture and surveillance. However, the inherent complexities and high volume of the data create the need for automation. For this purpose, we investigate the performance on three distinct datasets of seven supervised classifiers: Bagging, Random Forest, Extremely Randomized Trees, Stochastic Gradient Boosting, Extreme Gradient Boosting, the Support Vector Machine and 1-dimensional Convolutional Neural Networks.

To allow for a fair comparison between these methods, they have to be optimized in order to determine whether the differences in performance are due to their hyperparameter configuration or the inherent qualities of the algorithms. For this purpose we use Bayesian optimization, and compare its effectiveness with grid and random search. We find that even though Bayesian optimization gives a slight improvement in two cases, it has no impact on the ordering in quality of the models.

We find that Extremely Randomized Trees provides a good baseline. The Support Vector Machine is an excellent choice, as it achieves high performance while being both fast and easy to optimize. Tuning Convolutional Neural Networks is a time-consuming and unintuitive process, but once optimized they can provide good results.

1 Introduction

The hyperspectral data generated by modern airborne remote sensors contains information that allows for the detailed understanding and monitoring of the surface of the Earth. These sensors divide the area of interest into many small patches, and capture a spectrum at each of them. Spectra are characterized by hundreds of spectral channels or bands, each sensitive to a small window of the electromagnetic spectrum. The measurements are combined into an image of the surface, where every pixel contains a spectrum. The information in such datacubes make the precise identification and discrimination of different land cover classes possible. This has successfully been applied in domains such as precision agriculture, flood management, and the monitoring of protected natural areas or remote locations [75, 13, 42].

The sheer volume of the data however creates the need for automation. A solution is often found in the form of supervised machine learning models. The models need to be able to deal with the difficulties of working with high dimensional data, as well as numerous other challenges. These include working with an imbalanced training set, limited amounts of labeled data and a high number of classes. Dimensionality reduction, commonly used to address some of these difficulties, could lose some of the finer but critical relations for retaining the discriminability.

Over the last decades many classification algorithms have been explored, varying from linear regression to deep neural networks [54]. A popular class of models are decision tree ensembles. These are based on the idea that the accuracy of a decision tree can be increased by combining the results of multiple trees, while introducing one or more random elements in their construction. How this randomness is introduced varies between classifiers, and many examples, especially Random Forest, have been extensively analyzed in the literature [33, 78, 30, 13, 46]. Various neural network based classifiers have also been investigated, ranging from multilayer perceptrons to 1-dimensional convolutional or recurrent neural networks [53, 37, 52]. More recently, higher dimensional convolutional neural networks, that not only take the individual spectra but also the spatial context into account, have shown great promise [47]. In order to keep the scope of this work manageable these will not be considered, nor will we compare models where a CNN is solely used as a feature extractor. Another widely used model is the support vector machine (SVM), often reaching high scores while requiring little finetuning [30, 59, 61, 78, 19, 67].

We will examine 5 different tree ensembles and compare their effectiveness with the 1-D CNN and the non-linear SVM. In order to properly compare these models, we need to ensure they are used to the maximum of their capabilities. This comes down to optimizing the hyperparameters of the models. As the required expertise to find the optimal configuration manually will not always be available, there exist several approaches that aim to find the best settings automatically. We will analyze the performance of 3 such optimization algorithms: grid search, random search and Bayesian optimization.

The vast majority of the articles on hyperspectral classification we reviewed ei-

ther do not use such an automated approach for the hyperparameter tuning of their models, instead opting to tune their hyperparameters manually (e.g. [61], [59], [30], [44], [19], [53, 7]), or employ grid search (e.g. [75, 30, 78, 53, 45]).
For models that have low hyperparameter dimensionality and are robust to their settings, either manual tuning or grid search should be an adequate method for finding configurations that achieve near-optimal performance. For more complex methods, however, this is not necessarily the case, and approaches such as random search or Bayesian optimization (BO) may provide better results [5].
Remote sensing studies that use either random search or BO are more limited in number compared to grid search, and are typically applied in a slightly different context. For example, Acquarelli et al. [2] use a combination of grid and random search for spatial-spectral classification. Random search has been used in the context of classifying hyperspectral data to assess the quality of honey, crop classification with temporal multispectral data, and predicting the nutrient content in citrus leaves based on hyperspectral data [55, 81, 57]. BO has been applied to multispectral imagery in Sonobe et al. [70] and Su et al. [72], as well as for monitoring pigments in tea or classifying ink through hyperspectral data [71, 21]. We are not aware of studies directly applying either BO or random search to the pixel-wise classification of hyperspectral data that is generated by airborne remote sensors.
The contributions of this thesis are therefore twofold. First, analyzing the usefulness of classifiers such as Extreme Gradient Boosting and Extremely Randomized Trees, which have not yet been thoroughly investigated for classifying hyperspectral imagery. Second, assessing whether hyperparameter optimization algorithms such as BO are able to improve upon the performances reported in the literature of classifiers that were optimized either manually or with grid search.
The remainder of this work is structured as follows. In section 2 the seven models under consideration are described, followed by a description of the three optimization strategies in section 3. The details of the datasets used are given in section 4, and section 5 expands upon the design of the experiments. The results presented in section 6 are finally evaluated in section 7, and the conclusions are drawn in section 8.

2 Classifiers

Below a brief description of the inner workings of all seven classifiers is given.

2.1 Bagging

Bagging is a method that combines the results of many individually and independently trained models into a single outcome. Combining the results of these weak learners in the case of classification is for example done through a majority vote [9]. The weak learners used in this work are decision trees, and as such the whole model can be seen as a tree ensemble.

Given a training set of size k, every tree is trained on a different bootstrap sample of size k, drawn uniformly with replacement from the training set [9]. As the probability of a datapoint not being drawn is $1 - \frac{1}{k}$ and we draw k samples, the probability that a training point is not used in the building of a tree is $(1 - \frac{1}{k})^k$. Each tree is then built using a greedy algorithm that looks for the best split available at every node. The best split available is found by first calculating an impurity measure for every possible split, then selecting the split leading to the greatest gain in purity. We use the Gini impurity for the first 3 models, which at node v is defined by

$$I_G(v) = 1 - \sum_{i \in Y} p_i^2, \tag{1}$$

where p_i denotes the fraction of datapoints in node v labeled i, and Y is the set of all labels [23]. The change in Gini impurity is then calculated at a given node v by

$$\Delta I_G(v) = I_G(v) - \frac{n_i}{n_v} I_G(i) - \frac{n_j}{n_v} I_G(j), \tag{2}$$

with i and j the resulting child nodes of the split at node v, and n_x denotes the size of node x [23].
Other impurity measures, such as entropy, can substitute the Gini impurity in Equation 2. The differences in results when using entropy or the Gini impurity are typically small [64]. As opposed to entropy, there is no logarithm in the Gini impurity formula, hence it should be slightly faster to compute [65].

2.2 Random Forest

Random Forest (RF) introduces an extra random element to the bagging approach. At every node a sample of the features is drawn from all features without replacement. Instead of considering all possible splits, the split that will be performed at the node is the split maximizing the decrease in Gini impurity that is available in the feature sample [10]. How many features, or spectral bands in the context of remote sensing, are used to determine the split point is decided before the training of the RF starts.

2.3 Extremely randomized trees

The Extremely Randomized Trees (ERT) algorithm, unlike bagging and RF, does not use a bootstrap sample. Instead, it trains every tree on the whole training set. Similar to RF, at every node a sample of all features is drawn. Then, instead of finding the split that maximizes the gain in purity, a random split point is selected for each of these features. The splitting is then done by computing the purity gain for each of these randomly selected thresholds and selecting the best one. Trees are either grown to their maximum size, where all nodes contain a single datapoint, or the splitting halts once nodes reach a predefined size [29].

2.4 Gradient Boosting

Shifting from the previous models, where the results of many independently trained models are aggregated into one final prediction, we now consider another approach - boosting - where results from previous steps are taken into account during training. Here, after starting with an initial guess, models are fit sequentially to the data. At every step, the data is manipulated in some way to incorporate the information obtained in previous iterations. For example, the datapoints that were misclassified can become more likely to be sampled for the training set of the next iteration. These trained models can be any classifier, and we will again use decision trees.

The way that some datapoints are given more importance over others in the case of gradient boosting (GB), is by using the gradient of a differentiable loss function. The algorithm starts with an initial guess for each class. These odds can for example be set relative to the training class distribution. GB makes use of the logistic function to transform these odds to a scale where it can add the results of multiple weak learners, which would not be possible using probabilities [26].

At every subsequent step, a new tree is fit, where, instead of predicting a class label, the target values are the negative gradient of the loss function. The resulting predictions are added to the result so far, using a set learning rate. After all boosting iterations are complete, the final results are converted back to probabilities [26].

Gradient boosting with decision trees is a binary classifier, and to be able to use it we have to transform our multi-class classification task into multiple binary classification problems. This is done by a one-versus-all approach, where at every iteration a separate tree is built for each class. These predict the probability that a given sample will belong to that class. As our differentiable loss function we use the multinomial deviance.

2.5 Stochastic Gradient Boosting

Stochastic Gradient Boosting (SGB) adds a random component to the training phase of a GB model. Instead of using the full training sample, a different subsample randomly drawn without replacement from the training set is used to fit each decision tree. As fewer datapoints are used for the training process the computation time decreases, while the performance generally increases as the model became more robust to overfitting [27].

2.6 Extreme Gradient Boosting

Extreme Gradient Boosting (XGB) consists of a number of improvements over SGB, all related to further improving generalization capacity and preventing overfitting to the training data. Besides using a subset of the training datapoints, it allows for the subsampling of the features used in the building of the model, similar to RF. Furthermore, regularization is built directly into the loss

function, consisting of 2 parts. For a model M it is given by

$$L(M) = \sum_i l(\hat{y}_i, y_i) + \gamma T + 0.5\lambda ||\mathbf{w}||^2,$$

where the first term denotes the sum of the traditional loss function (e.g. multinomial deviance) over all training data points using the predicted labels $\hat{y}_i$ and the actual labels y_i. T represents the total number of leaves, $\mathbf{w}$ the leaf weights and γ and λ are the regularization parameters. The first regularization term penalizes the total number of leaves, effectively setting a threshold for the minimum improvement in loss that the splitting of a node has to exceed. The second term limits high values in the leaf nodes, in other words the contribution the prediction of a single leaf node can have on the final score (L2 regularization) [14].

In addition to these improvements to the model itself, the implementation of XGB, xgboost, includes a number of optimizations that make the algorithm considerably faster than (S)GB. This is especially the case on large datasets, among other things by using multiple cores to speed up the inherently sequential gradient boosting procedure [14].

2.7 Convolutional Neural Network

The one-dimensional convolutional neural networks considered here can be split into two parts. First, a set of alternating convolutional and pooling layers function as a type of feature extractor, given the input spectra as a feature vector. This is followed by one or more fully connected layers that use these extracted features to classify the input into one of the classes.

In the convolutional layers, a number of small windows or feature maps are shifted over the input which, in contrast to standard feed-forward neural networks and other methods considered here, are able to take the local spectral information of a band into account. As a result, neurons are only connected to a small number of neurons in the next layer, greatly reducing the number of trainable parameters and complexity of the model. The pooling layer then combines the information from these feature maps. A common approach is max pooling, which replaces a small region in the input by its maximum value, further decreasing the size and complexity of the features, as well as providing some invariance to translation [37].

The last pooling layer of the first part of the network is followed by one or more fully connected layers that act as the classifier, similar to a traditional multilayer perceptron. The last of these layers will have an equal number of neurons as there are classes in the classification problem, and through the softmax function their values can be turned into class probabilities.

During the training phase all the trainable weights between the neurons in the model are adjusted through forward propagation and error backpropagation. First a subset of the training data is presented as input to the model, resulting in an output vector with class probabilities for each datapoint in the batch. Then, the loss function with regards to the desired output (a probability of 1

for the class the datapoint belongs to and 0 for all others) is computed, and by using the partial derivatives of this loss function all trainable weights are updated, with the goal of decreasing the loss function. Once this process is done for all batches the training has completed one epoch, and will repeat itself until the specified number of epochs are complete [37].

2.8 Support Vector Machine

The support vector machine (SVM), initially designed as a binary classifier, aims to find the hyperplane best separating the input data. As the training data might not be linearly separable in its original space, it is transformed to a higher dimensional realm using the kernel trick [18]. In order to use the SVM as a multi-class classifier, the classification problem has to be converted to multiple binary sub-problems, similar to GB. There exist two common strategies; one-versus-one and one-versus-all. One-versus-one fits an SVM for every pair of classes, leading to a total of $\frac{n(n-1)}{2}$ models for n classes. One-versus-all creates a model for each class against all other classes, for a total of n classifiers for n classes. To aggregate the results for the one-versus-one approach, every model casts a single vote for the class it predicts, and the the final prediction is the class that has the most votes. In the case of one-versus-all, the class with the highest probability when paired against all other classes is predicted as the final label [8].

3 Hyperparameter Optimization

During training, machine learning algorithms typically operate by optimizing some criterion with respect to a number of internal parameters, such as the weights in a neural network or the support vectors of an SVM. In addition, these models often come with hyperparameters. These have to be set by the user before the training phase starts, as they cannot be estimated from the data. They control some aspect of the classifier that will affect the quality of the training. Examples are the number of layers of a CNN or the number of trees built in a RF [5]. We will refer to the model optimizing its inner criterion as the *training*, and to finding the optimal hyperparameter settings as the *tuning* of the model.

The number of hyperparameters that require tuning can vary greatly between models. For some models like bagging we will only consider 1 hyperparameter, for others such as XGB we optimize for 8. The CNN architecture space has the most considered in this work, with 16 values to tune. The hyperparameter dimensionality can be much higher; Bergstra, Yamins, and Cox [6] use a 238-dimensional search space to find an optimal image classification architecture. Projects like Auto-WEKA, that view the pipeline of feature extraction, model selection and hyperparameter optimization as one large optimization problem, can reach up to 768 dimensions.

The hyperparameters themselves come in different types and with different value

ranges. They can be continuous, such as the learning rate, discrete, like the batch size in a neural network or categorical, e.g. the choice of SVM kernel. Furthermore, some hyperparameters are conditional and are only active if another hyperparameter is set to a certain value. An example is the number of neurons in the nth layer of a CNN, which will only be active if the hyperparameter governing the number of layers is at least n.
The hyperparameter search spaces of different algorithms thus vary greatly in dimensionality, size and complexity. As their settings can greatly affect model performance, tuning them becomes an integral part to the proper comparison of classifiers. After first describing grid search we look into two alternatives for the automatic optimization of hyperparameters; random search and Bayesian optimization (BO).

3.1 Grid search versus Random search

In order to use grid search the user has to specify a set of values for each hyperparameter. Then, all possible combinations of these hyperparameters will be evaluated. The total number of configurations evaluated is therefore the product of the cardinality of each set [5].
While grid search is a reliable option for models with low dimensional search spaces, exhaustively searching all combinations in higher dimensions quickly grows infeasible, due to the exponential growth of the number of configurations. Furthermore, for models such as neural networks only a subset of the hyperparameters have a big impact on the performance, and which hyperparameters have a big impact is not consistent between datasets. Hence, while the search space may be high dimensional, it has a low effective dimensionality [5].
To illustrate why grid search will be ineffective in cases with low effective dimensionality, consider the example in Figure 1. If both parameters were equally important having a uniform grid over the 2 dimensional search space would be ideal, as it leaves no region disproportionately unexplored. However in this case, where there is an imbalance in importance between hyperparameters, a number of trials are wasted. As can be seen in Figure 1, with grid search the function will only be evaluated at 3 different values for the important hyperparameter, even though 9 configurations have been computed; the uniformly distributed points in 2-dimensions of the grid search provide an inefficient coverage of the one dimensional subspace of the important hyperparameter. Note that the unimportant hyperparameter has a similarly inefficiently covered subspace, and as a result it does not matter which hyperparameter is the important one.
The opposite to grid search would be to evaluate 9 configurations uniformly spaced along a diagonal. Both 1-dimensional subspaces will be uniformly covered with 9 trials, thus likely finding a better solution in the case where one hyperparameter accounts for (nearly) all the variance of a function. In the balanced case however, as the search is missing evaluations in large portions of the inefficiently covered 2-dimensional search space (in the corners opposite the diagonal), the solution found is likely worse than with grid search.

Random search evaluates randomly chosen, independent trials, and falls in between the 2 extremes of grid search and evaluating points on the diagonal. As can be seen in the right panel of Figure 1, the subspace of the important parameter is covered with 9 randomly distributed trials instead of 3, while still having a decent coverage of the 2-dimensional space. In the imbalanced case of Figure 1 this leads to finding a better configuration.
These problems worsen in higher dimensions for grid search. The number of combinations will grow exponentially with the number of dimensions, and with more hyperparameters an exponentially growing number of trials will become irrelevant. Random search on the other hand does not suffer from this problem; regardless of how many irrelevant dimensions are present, the coverage of the subspaces of the important parameters will stay the same given the same number of trials.
To illustrate, if we were to add another completely irrelevant dimension to Figure 1, again allowing for 3 possible values, grid search would evaluate a total of 27 configurations. Of these 27 only 3 will be unique values for the single important hyperparameter, whereas random search would evaluate 27 distinct values in 27 trials. In the case where 2 out of 3 hyperparameters have a significant impact, grid search will evaluate 9 uniformly distributed configurations in the relevant 2-dimensional subspace out of the 27 trials, in contrast with random search, which evaluates 27 randomly distributed configurations in the same relevant 2-dimensional subspace.
As in practice often only a subset of the hyperparameters account for a large portion of the variance in performance, random search should often provide better results compared to grid search. However, the results may be less consistent when the total number of evaluations is low [5, 38].
Random search has some other practical benefits over grid search, while being equally easy to implement. As all individual trials are independent, new machines can be added to a running experiment on the fly, failing machines can be removed without problems and the experiment can be stopped at any point without missing evaluations in a certain portion of the search space [5].

3.2 Bayesian optimization

Both grid search and random search do not incorporate the results of previous evaluations when determining where in the search space the next trials will be placed. This ignores valuable information on where the most promising regions might lie. In contrast, Bayesian optimization (BO) treats the tuning of hyperparameters as the sequential optimization of a black-box function, where by using Bayes' theorem all previous observations are included in the decision process of what configuration to evaluate next [40].
The black-box function we wish to optimize in our case takes a set of hyperparameters and returns the accuracy (or any other metric by which we wish to optimize) of the classifier trained with those settings. In the case of complex models such as deep neural networks the evaluation of this function, which would involve training the model and evaluating its performance to obtain the

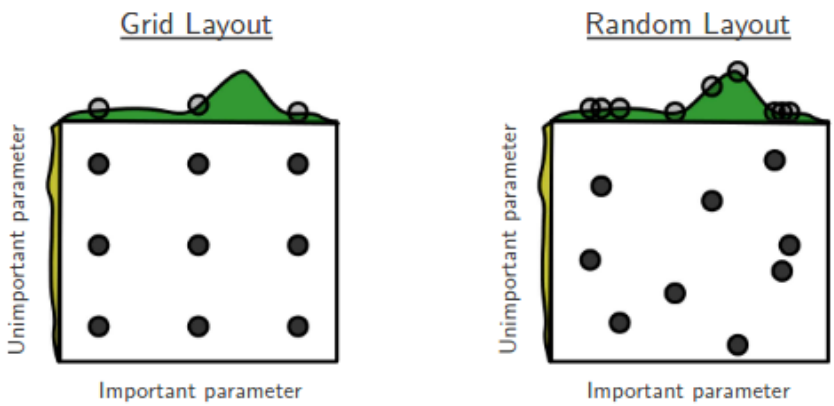


Figure 1: Grid search and random search trials for optimizing a function defined as $f(x) = g(x) + h(x)$. $g(x)$ is shown at the top in green and $h(x)$ left of the box in yellow [5].

accuracy, can take a long time. Combined with the fact that we often cannot obtain the derivatives of the black-box function, i.e. the derivatives of the accuracy with respect to the hyperparameters, applying function optimization techniques such as gradient descent to find the optimal hyperparameter configuration directly is impossible [12].

Instead, BO constructs a probabilistic model of the objective function, where combinations of hyperparameters are mapped to a probability distribution over accuracy values. This model is called the *surrogate function*. Common choices for this surrogate function include Gaussian Processes (GP), Random Forests (RF) and Tree Parzen Estimators (TPE) [22].

Even though an RF is generally not used as a probabilistic model, it can be turned into one by returning a Gaussian probability distribution $\mathcal{N}(\mu_\lambda, \sigma_\lambda^2)$ for the set of hyperparameters λ, where μ_λ is the average prediction of all trees in the forest and σ_λ^2 its variance [73].

There are a number of factors to take into account when choosing the surrogate function. For relatively low dimensional search spaces with numerical hyperparameters (Frazier [25] mentions in cases with less than 20 dimensions) GP-based implementations typically outperform the tree based models, such as RF and TPE [38]. While discrete variables can still be used relatively easily by relaxing them to be continuous and rounding them to the nearest integer at the end, GPs have more trouble with categorical and conditional variables [20]. Categorical values can be dealt with for instance by using one-hot encoding [28]. One solution for conditional parameters is using separate GPs for groups of variables that are active at the same time [4]. This does however lose sight of the dependencies between groups [69]. Generally however, when dealing with categorical or conditional variables, or when there are many hyperparameters to consider,

the tree based approaches are favored. This is a result of the properties of trees that make them inherently able to deal with conditional and categorical values, as well as being faster than GP regression in higher dimensions [38].
To determine where the next evaluation of the objective function will be, an *acquisition function* is used on the surrogate function to find regions of high interest. This acquisition function tries to find a balance between exploration and exploitation. Exploration focuses on those regions of the search space where the uncertainty is high (high variance), whereas exploitation favors regions where the expected performance is high (high mean).
A commonly used acquisition function is the Expected Improvement (EI) [40]. It is defined by

$$EI(\mathbf{x}|D) = \int_{f_{best}}^{\infty} (y - f_{best}) p(y|\mathbf{x}, D) dy, \tag{3}$$

where y denotes performance, $\mathbf{x}$ a possible combination of hyperparameters to evaluate next and D the set of hyperparameter and performance pairs evaluated so far. f_{best} represents the best score found at the current time. Note that $p(y|\mathbf{x}, D)$ is the surrogate function; the probabilistic model of the objective function based on the evaluations made.
Acquisition functions like EI are computationally inexpensive to evaluate and can be calculated analytically. The problem of finding the optimum of a function has now shifted, from optimizing the original black-box function to optimizing the acquisition function on the surrogate function. How this is done depends on the implementation details. Common algorithms for this optimization include Dividing Rectangles, Covariance Matrix Adaptation Evolution Strategy and Monte Carlo sampling [12, 4].
This process of computing the surrogate function, finding the next point to evaluate through maximizing the acquisition function, evaluating said sample point and using this observation to update the surrogate function makes up the main loop of BO.
Bayes' theorem states that the posterior probability of a model M with evidence E is proportional to the product of the prior probability of M and the likelihood of E given M [12]:

$$P(M|E) \propto P(E|M)P(M). \tag{4}$$

Applied to our optimization problem, the prior is represented by the surrogate function. Upon evaluation of a new sample point the results are added to the evidence, and the surrogate function is updated to become the posterior distribution. This will act as the prior in the next iteration of the process.
Pseudocode for the whole BO procedure is given in Algorithm 1. An illustration of the main loop in the 1-dimensional case using a GP as the surrogate function can be found in Figure 2.

A disadvantage of BO is that some choices have to be made for the process itself, which in some sense only shifts the problem of tuning hyperparameters, from the model to the optimization algorithm. These include the choice of acquisition function, surrogate function and the total number of iterations. Other choices that have to be made include where to locate the initial trials, and

Algorithm 1 Main loop for Bayesian optimization of an unknown function f [25][35]

$\{\mathbf{x}_n, y_n\}_{n=1}^{n_0} \leftarrow$ evaluate n_0 initial trials, where $y = f(\mathbf{x})$
$M_{n_0} \leftarrow$ fit the surrogate function using the initial evaluations
for $n = n_0 + 1, ..., N$:
    $\mathbf{x}_n \leftarrow$ maximum of the acquisition function over M
    $y_n \leftarrow$ evaluation of objective function using $f(\mathbf{x}_n)$
    $M_n \leftarrow$ update the surrogate function with the new observation $\{x_n, y_n\}$
Return the best configuration found

how many should be evaluated before starting the optimization loop. Options for this initialization include random sampling, Latin hypercube sampling and quasi-random sampling with Sobol sequences [35]. If previous knowledge, such as the results from a random search experiment, are available these can be used as the basis for the optimization instead. In the case where the model has already been optimized on different datasets, meta-learning can be employed to use well performing configurations on the most similar datasets as the initial trials for the new problem [24].

3.3 Local Penalization

While the BO process is inherently sequential, it can be parallelized by evaluating a batch of parameter settings instead of only evaluating the maximum of the acquisition function. The most naive way is to fill the batch with random configurations. In González et al. [31] a heuristic to model the interactions between evaluations is proposed, with the goal to facilitate a good batch design. This process is called local penalization (LP), visualized in Figure 3. It is based on the observation that the main effect on the acquisition function of a function evaluation will happen in its neighbourhood, and thus the acquisition function for subsequent parameter choices in the current batch should be less likely to lie therein. Latin hypercube sampling is recommended as the initial design strategy to use with LP if no previous evaluations are available.
If we want to place n Latin hypercube samples, the range for each variable is first partitioned into n intervals of the same size. Then, the samples are placed in such a way that when projecting onto any single dimension there will be exactly one sample placed in every interval [50]. In the 2 dimensional case this results in there being exactly one sample in each row and each column. Note that solely evaluating Latin hypercube samples can already be seen as an optimization strategy, that falls in between grid and random search. No part of the search space is disproportionately unexplored, and it does not suffer from the issues of grid search where the subspaces are inefficiently covered. In Bergstra and Bengio [5], however, they found that this approach was no more efficient than the expected results with random search.

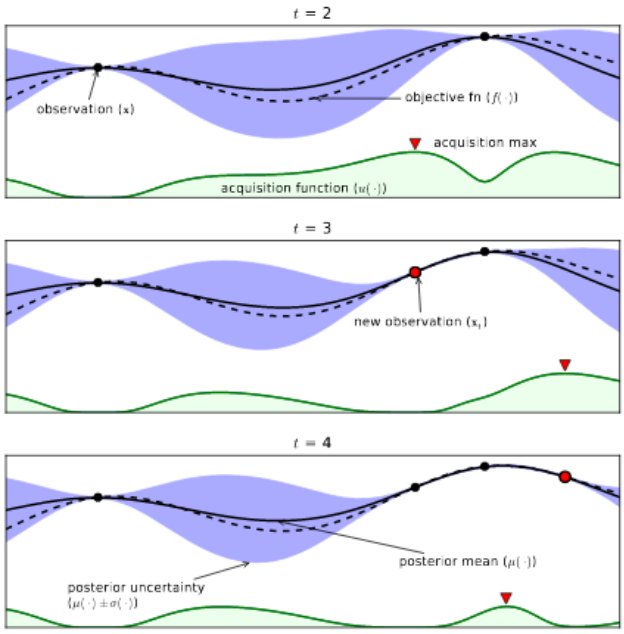


Figure 2: 3 steps of BO on a 1-dimensional toy example, where the black-box function is given by the dashed line. The solid black line represents the mean of the Gaussian Process, with the variance shown in blue. The acquisition function is shown in green. The red triangles indicate the maximum of the acquisition function at each step, and the dots indicate black-box function evaluations [12].

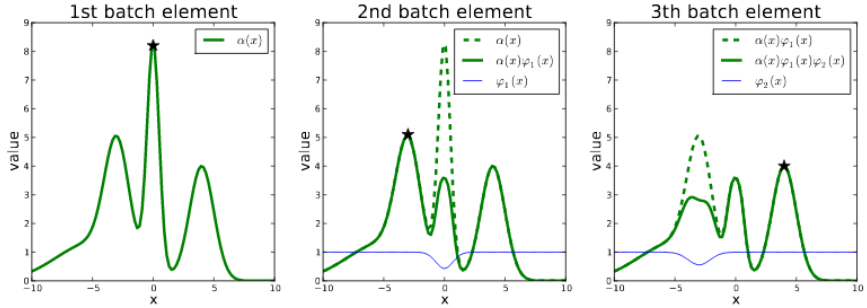
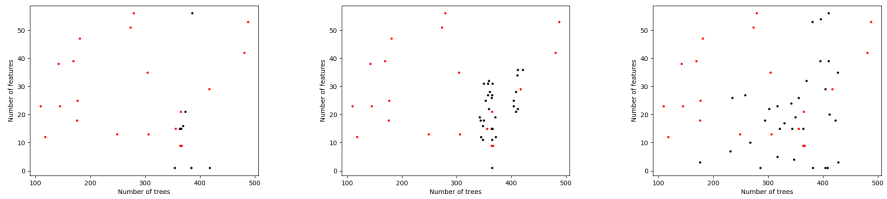


Figure 3: Local penalization for a batch of size 3. Black stars represent the acquisition function ($a(x)$) maxima that will be included in the batch. $\phi_n(x)$ illustrates the local penalization by which the acquisition function was multiplied to obtain the acquisition function for batch element $n+1$.

The effect of using LP and that of different batch sizes is illustrated in Figure 4. Every dot in the images represents one model evaluation for a total of 50 evaluations per image, where the first 20 are initialized using Latin hypercube sampling (shown in red). In the case without LP the BO remains focuses on the bottom right, and the rest of the search space remains unexplored after the initial configurations. With a batch size of 2 there is a less dense concentration of points in the bottom right and a more expansive exploration. Further increasing the batch size exaggerates this behaviour. Out of the three runs shown in the images the run with a batch size of 4 found a better model than when using a batch size of 2, which in turn outperformed the case without LP.



No LP | LP with batch size 2 | LP with batch size 4

Figure 4: Different BO runs with a total of 50 evaluations each. The 20 initial points sampled with Latin hypercube sampling are shown in red.

3.4 Hyperparameter Importance

Relying on one of the previously discussed optimization strategies leaves the user without insight into which hyperparameters contributed the most to the overall performance, and in what way. To this end, the probabilistic model used

during BO cannot only be used for the optimization process, but also to get a sense of the effect and importance of individual hyperparameters and their interactions after the optimization is complete.
In Hutter, Hoos, and Leyton-Brown [38] the authors developed an algorithm that computes marginals from a RF surrogate function in linear time. Recall that the surrogate function maps a configuration of hyperparameters to a probability distribution over performance, hence the marginals give insight into the effect of a subset of the hyperparameters on the performance. The marginals are calculated by averaging over all instantiations of the hyperparameter(s) not included in the marginal [38].
These marginals can in turn be used with a functional analysis of variance (fANOVA) to determine the effect of individual hyperparameters as well as their interactions on the overall variance in performance. fANOVA takes a black-box function and decomposes its variance into additive components, where the components are all the possible subsets of the input variables. The components in our case consist of all possible combinations of any length of input hyperparameters. With the fANOVA we then know the contribution of every component to the total variance, and we can quantify the importance of a component by the fraction of the total variance that it explains [38]. Important to note is that the results from the fANOVA heavily depend on the chosen ranges. If an otherwise crucial parameter is only allowed to vary between values where its impact on the performance is relatively constant, the fANOVA will quantify it as unimportant. Whether these are good or bad settings is irrelevant. As such the fANOVA results should only be seen as information about the optimization procedure that was run with the specified ranges. An illustration of the dependence on the ranges is given in subsection 6.3.
These insights do not have to be limited to just the effect of the hyperparameters on the accuracy; any measure for which we have data can be used. For example, the effect different settings have on the time it takes to fit the model can be investigated instead.
We use the implementation made by the authors of Hutter, Hoos, and Leyton-Brown [38] [1].

4 Datasets

The classification and optimization algorithms are compared using three different datasets, described below. We use the Indian Pines dataset as it is widely used for evaluating classifiers on remotely sensed hyperspectral imagery (e.g. [30, 37, 53, 78, 77]). This provides a confirmation that our pipeline functions as expected, by verifying that when using the same settings our performance for widely used classifiers is the same as what is reported in the literature. Additionally, it serves as an opportunity to compare the usefulness of lesser used models like ERT and XGB with those widely used classifiers. As Indian Pines does not have a designated training and testing set, differences in performance

[1] `https://github.com/automl/fanova`

Table 1: Class distribution in the Indian Pines dataset.

Index	Class	Number
A	Alfalfa	46
B	Corn-notill	1428
C	Corn-mintill	830
D	Corn	237
E	Grass-pasture	483
F	Grass-trees	730
G	Grass-pasture-mowed	28
H	Hay-windrowed	478
I	Oats	20
J	Soybean-notill	972
K	Soybean-mintill	2455
L	Soybean-clean	593
M	Wheat	205
N	Woods	1265
O	Buildings-Grass-Trees-Drives	386
P	Stone-Steel-Towers	93
-	Sum	10249

across experiments can in part be attributed to what labeled samples were used during training. For this reason, we also perform experiments on the University of Houston image, where the training/testing split was made beforehand. Finally, for the Lunar Crater Volcanic Field dataset we use the data and labels as in Merényi et al. [52] to allow for a fair comparison with the self-organizing map-hybrid artificial neural network (SOM-hybrid ANN) model described there, which has not been compared with other modern classifiers. The way the performance of the models will be evaluated is described in section 5.

4.1 Indian Pines

The first image used was acquired on June 12, 1992 using the Airborne Visible / Infrared Imaging Spectrometer (AVIRIS) sensor [49]. It consists of 145x145 pixels, with 220 spectral reflectance bands. After removing 20 water absorption bands in the ranges [104, 108], [150,163] and 220, the remaining 200 bands are used as features for the classification, as in Ghamisi et al. [30]. A total of 10249 pixels divided into 16 classes have been labeled. The description and distribution of the classes can be found in Table 1.

4.2 University of Houston

The second dataset was acquired in 2012 by the Compact Airborne Spectrographic Imager with a size of 349*1905 pixels and 144 spectral radiance bands

Table 2: Class distribution in the University of Houston dataset.

Index	Class	Train	Test
A	Healthy grass	198	1053
B	Stressed grass	190	1064
C	Synthetic grass	192	505
D	Trees	188	1056
E	Soil	186	1056
F	Water	182	143
G	Residential	196	1072
H	Commercial	191	1053
I	Road	193	1059
J	Highway	191	1036
K	Railway	181	1054
L	Parking lot	192	1041
M	Parking lot 2	184	285
N	Tennis Court	181	247
O	Running Track	187	473
-	Sum	2832	12197

[1]. Unlike the Indian Pines (IP) image it has a designated training and testing set, with a total of 15029 labeled pixels split into 15 classes. Details are given in Table 2. The mean spectral signatures for each class are given in Figure 5. This illustrates the difficulty of hyperspectral image classification; the spectra overlap and cross in many places and the differences between them can be very subtle.

4.3 Lunar Crater Volcanic Field

The Lunar Crater Volcanic Field (LCVF) image was again made by the AVIRIS sensor, taken on April 5, 1994. From the 224 bands present in the original image, 194 are used after removing noisy and overlapping bands. Additionally the image has been atmospherically corrected and converted to reflectance units. We received a private copy of the data and labels in a preprocessed state as described in Merényi et al. [52]. The size is 614x420 pixels, 5274 of which were manually labeled into 23 classes, as shown in Table 3. We use the dataset before the augmentation done in Merényi et al. [52], where all classes were expanded to include at least 14 samples. As a result, in this work class G is 7 samples smaller, and classes J and V are 2 samples smaller.

To further illustrate how similar the spectra can be, in Figure 6 we plot the training spectra per class pulled apart. For classes such as Q, R, S and T the spectra are visually nearly identical, only separated by a small margin on the y-axis.

Table 3: Class distribution in the Lunar Crater Volcanic Field dataset.

Index	Class	Train	Test
A	Hematite-rich cinders	72	309
B	Rhyolite of Big Sand Spring Valley	22	200
C	Alluvium #1	50	200
D	Dry playa	160	210
E	Wet playa #1	115	210
F	Young basalt	21	40
G	Shingle Pass Tuff	7	105
H	Alluvium #2	50	220
I	Old basalt	36	100
J	Dense scrub brush stands	12	250
K	Basalt cobbles on playa	37	248
L	Ejecta blankets #1	78	214
M	Alluvium #3	14	200
N	Dry wash #1	15	300
O	Dry wash #2	54	310
P	Dry wash #3	45	298
Q	Wet playa #2	15	220
R	Wet playa #3	14	50
S	Wet playa #4	15	59
T	Wet playa #5	18	64
U	Alluvium #4	36	200
V	Wet playa #6	12	50
W	Ejecta blankets #2	33	203
-	Sum	931	4260

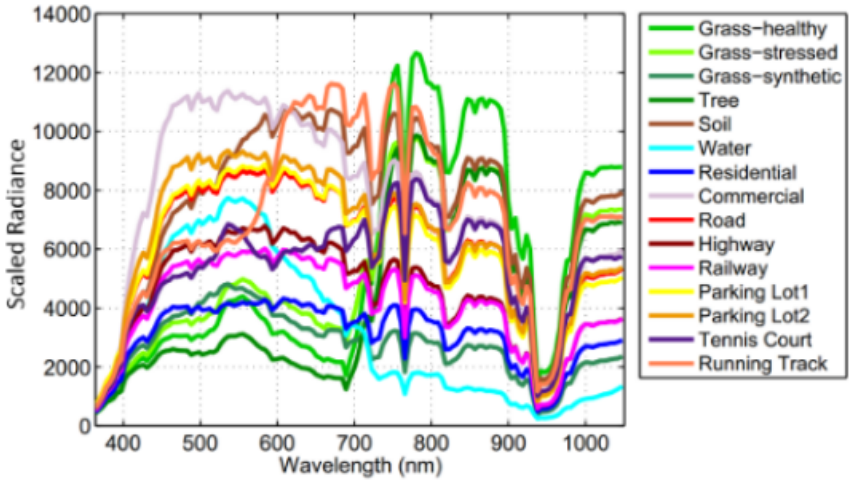


Figure 5: Mean spectral signatures for all 15 classes of the UH image [77].

5 Experiment Design

Considering all possible values for every parameter would lead to a huge number of possible combinations, where the vast majority is not worth evaluating. As such, ranges in which the parameters of the models are expected to give good performance have to be specified beforehand. In this section we examine the recommended parameter ranges for the models we will use, followed by a description of what experiments are ran on the individual datasets. In Table 4 an overview of the ranges is given.

For bagging, RF, ERT, SGB and SVM we use the scikit-learn package, implemented in Python [60]. For XGB the xgboost Python package is used [14]. The CNN models are made with Keras [16].

As mentioned before, BO comes with a few of its own hyperparameters. We keep these at fixed values to avoid the problem of separately optimizing another layer of parameters. Given the hyperparameters we consider, all methods except CNN lack categorical or conditional variables and have a dimensionality less than 20, hence for these models we use the GP based GPyOpt [3]. We use the local penalization strategy with a batch size of 4 to speed up the process, and the commonly used EI as the acquisition function. For the initial configurations we use Latin hypercube sampling, where we try to adhere to the practice of Jones, Schonlau, and Welch [40] where $10k$ points are initialized with k the dimensionality of the hyperparameter space. Using Latin hypercube sampling is also recommended by González et al. [31]. If the computational budget allows, we allocate around 40% of the total number of evaluations to this initial design. This follows from an experiment in Brochu, Cora, and De Freitas [12] where, in a 15-dimensional search space with 30 Latin hypercube samples, performance

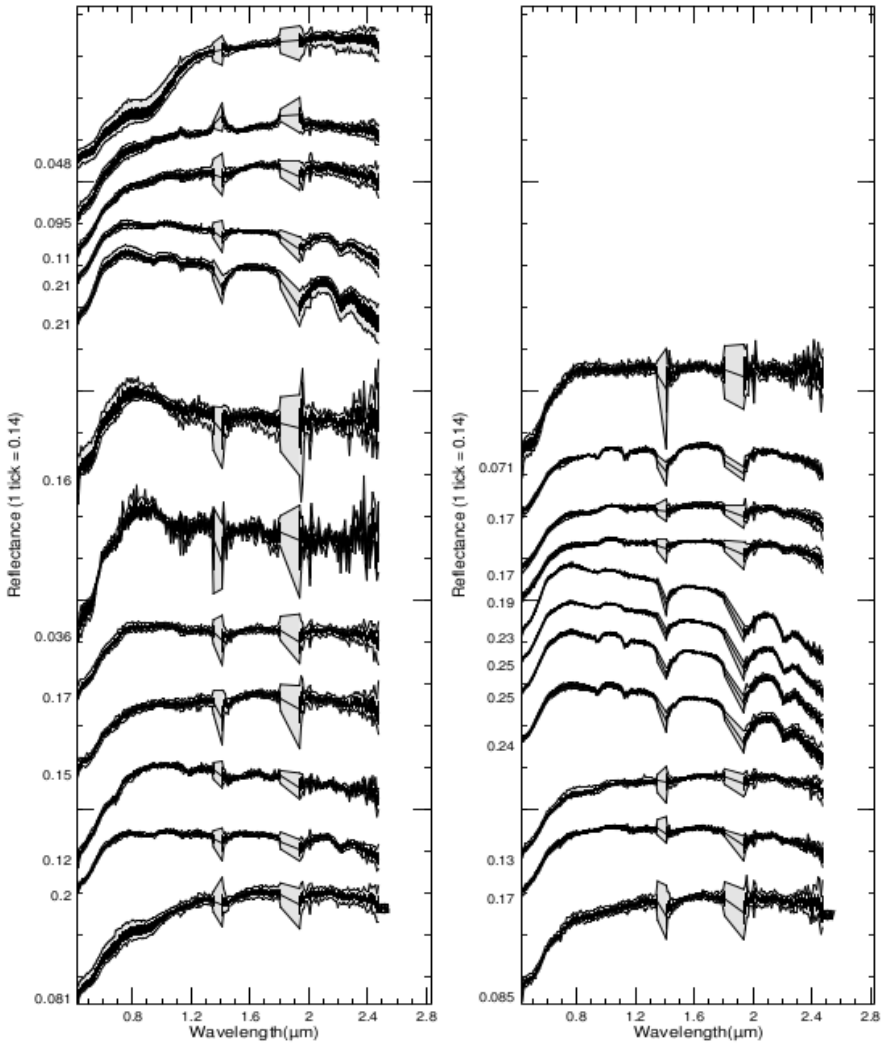


Figure 6: Mean (black line), standard deviation (vertical bars) and min/max value (grey area) of the training LCVF spectra per class. The spectra are in alphabetical order from top to bottom, left to right.

plateaued after around 80 epochs. Feurer, Springenberg, and Hutter [24] used between 12.5% and 62.5% initial samples but did not find a single best ratio.
As we are dealing with conditional variables for the CNN architecture search we use a TPE based implementation of BO, named hyperopt, with the default settings. The number of initial samples does not have to be specified when using hyperopt [6].

5.1 Bagging

The one choice that has to be made when using the bagging algorithm with decision trees is how many trees will be constructed. In Breiman [9] it was found that after a certain point the increase in performance becomes negligible, but no significant drop was reported at any point afterwards. How soon the performance converges depends on the dataset. In one experiment, using 25 trees led to the misclassification rate being only 0.1 percentage point lower than when using 50 or 100 trees. 100 trees was the highest value considered for the number of trees [9]. By preliminary exploration on the IP image we found that using 200 trees resulted in the same performance as 500 (80.45%). As there does not seem to be an indication that any performance will be gained from setting the upper bound even higher, we set the upper bound to 500 and trace this hyperparameter in the discrete range $[25, 500]$.

5.2 Random Forest

The 2 parameters to consider are the number of trees to be built and the number of features to consider when finding the optimal split. In Ham et al. [33] forests larger than 100 trees did not improve the performance. Oshiro, Perez, and Baranauskas [58] suggest between 64 and 128 trees and found that there was no significant increase in performance from 128 to 4096 trees. We adopt 100 trees as the lower bound in our search. As we wish to include the models used in related works like Ghamisi et al. [30] and Mou, Ghamisi, and Zhu [53], where 300 and 200 trees were used respectively, the upper bound has to be at least 300. Similar to bagging, RF does not seem to overfit, but after a certain number of trees (how many differs between problems) results stop improving significantly [10]. Preliminary results suggest that differences between using 300 and 500 trees are minimal. Using 13 features, for example, with 500 trees achieves a testing accuracy of 81.84% on the IP data while 300 trees reached 81.85%. We once again adopt 500 as the upper bound.
The number of features can be as low as 1, where a random feature is selected to split on at every node. As training with only a single feature is very fast, this is taken as the lower bound. The optimal number of features for classification is often reported as the square root of the dimensionality of the input [30]. In the experiments in Breiman [10] the error did not change much when varying the number of features over a wide range, but to allow for a wide exploration we take 4 times the recommended value as the upper bound. This comes down to 56 for IP and LCVF and 48 for UH.

We can adapt the size of the sets for the different hyperparameters when using grid search to mitigate its problem with imbalanced hyperparameter importance. As the number of features should have considerably more impact on the performance ("The number of prediction variables is referred to as the only adjustable parameter to which the forest is sensitive" [30]) than the number of trees, we can sample more points along that dimension. Our preliminary results suggested that differences between 300 and 500 trees are minimal, and we will sample the number of trees from the set $\{100, 300, 500\}$. The rest of the budget is then devoted to uniformly spaced evaluations for the number of features.
Contrary to the original way of combining the decision trees, the scikit-learn implementation used in our experiments combines classifiers by averaging their probabilities, instead of letting each tree vote for a single class. This probability is given by the fraction of samples of a class in a leaf node.

5.3 Extremely Randomized Trees

The recommendations for both the number of trees and the number of features considered at each split for ERT are the same as for RF; the more trees are grown, the higher the accuracy will likely be, and the optimal number of features is approximately the square root of the input dimensionality. While the convergence of ERT is slightly slower than that of RF and bagging, 100 trees was large enough to ensure convergence on all datasets evaluated in Geurts, Ernst, and Wehenkel [29]. We use the same ranges for both the number of trees and the number of features as with RF.
As for the minimum samples a node has to contain in order for it to be split, the default and lowest possible value of 2 was found optimal for all 12 classification problems in Geurts, Ernst, and Wehenkel [29]. In one experiment the optimal value increased to 7 when 10% of the labels were randomly flipped. They conclude that the optimal value increases depending on the presence of label noise, as stopping the splitting of nodes earlier reduces the effect of individual training samples. Our datasets should not have this much mislabeled data. Nonetheless, we use 7 as the upper bound for our search, as it might counteract other types of noise present in the images.

5.4 Stochastic Gradient Boosting

With SGB, more trees can lead to a decrease in performance, as it is not immune to overfitting. Furthermore, the number of trees built is directly related to the learning rate used, as with a lower learning rate more trees will be needed for convergence. Experiments with standard GB in Friedman [26] found that for learning rates lower than 0.125 increases in performance on held out test data stagnated. Generally, a lower learning rate resulted in an increased generalization capacity, albeit with diminishing returns for smaller values like 0.06 and 0.03.
The recommendation in Friedman [26] is to choose the number of trees as large as computationally convenient, followed by tuning the learning rate such that

the accuracy reaches its maximum close to the chosen number of trees. In experiments done in Friedman [26] they found it can take up to 500 iterations to converge. Similar to the previous methods however nearly optimal performance on a held out test set is achieved rapidly ($\pm$100 iterations), followed by a levelling off of the curve. We will follow the recommendation by fixing the number of trees to be 500, as well as the empirical evidence that learning rates smaller than 0.1 lead to the best performance [27]. Friedman [27] uses a learning rate of 0.005 on small datasets containing 500 samples which we will use as the lower bound. On larger datasets (N=5000) the learning rate was increased to 0.05. He did not vary this depending on the choice of maximum depth or subsample ratio. We account for possible interactions with these hyperparameters by increasing the upper bound to 0.15.
The subsample size, i.e. the randomly sampled fraction of the training set used for each tree, tends to be optimal between 0.5 and 0.8 for classification [27]. We set our lower bound to 0.4 and the upper bound to 1.0 (standard gradient boosting). No subsampling turned out to be optimal for a small number of the classification problems in Friedman [27].
Lastly, we have to specify the maximum depth the individual decision trees are allowed to grow to. We follow the recommendation in Hastie, Tibshirani, and Friedman [34] and use the range $[4, 8]$. Note that the subsample size and the maximum depth are also related: allowing bigger trees will increase the risk of overfitting, which in turn can be combated by decreasing the subsample size.
The scikit-learn implementation sets the initial guesses relative to the class priors.

5.5 Extreme Gradient Boosting

Being the youngest and one of the more complex classifiers considered in this work, usage recommendations for XGB are somewhat less clear. There exist some recent studies unrelated to remote sensing that use optimization algorithms in combination with XGB, although there seems to be no consensus on which hyperparameters to optimize for [80, 63, 74]. We optimize for the same hyperparameters plus lambda as in Zhang et al. [80], and add the minimum purity and gamma compared to Putatunda and Rama [63]. The ranges used encapsulate the settings used in the (limited) number of studies we found that use XGB on hyperspectral imagery (though these are all different in some way, for example used after feature extraction with a CNN, or on data acquired from a hyperspectral sensor mounted on a tripod)[7, 39, 48].
All the hyperparameters present in SGB are used for XGB, some with slightly different usage recommendations. Generally, the learning rate can be somewhat higher with the increased regularization. To illustrate, the default learning rate for the xgboost implementation of XGB is 0.3, which we now use as the upper bound, while the scikit-learn SGB implementation has a default learning rate of 0.1. In any case, the optimal value again depends on the other settings such as the number of trees built, which we fix to 500 for the same reason as with SGB. For both the maximum depth and the subsample ratio we also use the same

ranges. These have the goal to prevent overfitting and decreasing computation time when lowering their values, at the risk of a lower generalization capacity when set too low. The optimal maximum tree depth for XGB was found to be as low as 1 in Zhang et al. [80] and as high as 16 in Putatunda and Rama [63], hence $[1, 16]$ is the range we will use.
Four more parameters are introduced, all related to regularization. The column (bands) subsample ratio fulfills a similar purpose as the row (pixels) subsampling ratio, and we trace it in the same range of $[0.4, 1]$. The minimum purity a node needs to have in order to be split tends to have a lower bound of 0, its lowest possible value. The upper bound varies, for instance set to 4 in Xia et al. [79] and 10 in [74]. We adopt the higher bound of 10. Gamma, the variable that stops the growing process once a partition will be unable to meet a certain purity gain, has varying ranges across articles. The lower bound is again usually the lowest possible value of 0. Some articles set the upper bound very close at 0.01 or 0.02 [74, 79]. In Zhang et al. [80] the optimal value found was 1.099. We allow it in the range $[0, 1.5]$. In Putatunda and Rama [63] the lambda parameter has a lower bound of 0, which is also the theoretical lower bound, and an upper bound of 1, which is the default value in the xgboost implementation.

5.6 Convolutional Neural Network

We divide the tuning of the CNN into two phases; first finding an optimal architecture, then fully training it to achieve its maximal potential.

5.6.1 Architecture search heuristics

In Saxe et al. [68] it was found that a large fraction of CNN performance can be attributed to just the feature extraction part of the architecture initialized with random weights, without training the network through backpropagation or unsupervised pretraining. The architecture in this case consists of the properties of the convolutional layers such as the feature map and pooling sizes. Consequently, an efficient way to find promising architecture candidates in the vast search space is to optimize based on the performance achieved when classifying features extracted with almost or completely random weights [68, 32].
The way the random weight performance of a model is calculated in Saxe et al. [68] is by using a linear SVM with a small grid search of 3 values ($\{10^{-3}, 10^{-1}, 10)$ to classify the random features, and averaging the best performance over a number of random initializations. They found that when the random weight performance was high, so was the fully trained performance and vice versa.
Hahn et al. [32] add the fully connected layer to the architecture, and their heuristic involves first training the convolutional layers for a small number of epochs (e.g. 3), and then the fully connected layer for some more (e.g. 30). Performance is averaged over a number of random initializations to give the final heuristic score. A BO loop is then used to optimize for the architecture hyperparameters.
We test three architecture search heuristics based on these studies. The first

we will consider combines the two by using completely random weights in the convolutional layers followed by training the fully connected layer for 30 epochs. Secondly, we adopt the approach of Hahn et al. [32] entirely by training the feature extraction part for 3 and the classifier for 30 epochs. Finally, we use the Saxe et al. [68] approach to determine random weight performance and add a BO loop to optimize for it.

As for the hyperparameters, we look at models ranging from 1 to 5 convolutional layers deep. This includes 1 layer models used in remote sensing papers such as Mou, Ghamisi, and Zhu [53] and Hu et al. [37], as well as 4 layer networks like in Wu and Prasad [77] or 5 layers as in Ghamisi et al. [30]. In the convolutional layers we consider filter sizes in the range $[3, 25]$, again including the previously mentioned papers, and allow for between 5 and 100 convolutions in a layer, as well as max pooling of sizes between 2 and 5 for identical reasons. We decrease the lower bound for the pool size to 1 (no pooling) in the 5th layer for two reasons: to include the IP network of Ghamisi et al. [30] where the 5th layer uses pooling of size 1, and to increase the number of valid deep architectures.

Initial experiments showed that out of the 3 values tested for C in the grid search of Saxe et al. [68], the largest value always gave the best performance. For this reason we remove the lowest value for C and replace it with 10^3, after which both 10 and 10^3 at times occurred as the best value for C.

Hahn et al. [32] include the fully connected layers into the architecture, and for the number of neurons in the first fully connected layer we consider values between 64 and 512 neurons, again incorporating models from the literature such as the layer with 100 neurons in Hu et al. [37] or 256 neurons in Ghamisi et al. [30]. The second layer always has a size equal to the number of classes as it will be used to assign class probabilities.

5.6.2 Training hyperparameters

Upon finding a promising architecture that performed well in the heuristic, the hyperparameters specific to the training process can be optimized separately (although this split is not always trivial, and parameters such as dropout could be argued to belong to either). For this we again use a BO loop, optimizing for the number of epochs, the batch size and values relating to the Adam optimizer, such as the learning rate. We use Adam over standard stochastic gradient descent (SGD) for two reasons. First, its learning rate and other hyperparameters in general require little tuning [43]. According to Ruder [66] the default learning rate often achieves the best results, which also makes it a good choice to use with our heuristic based on Hahn et al. [32]. Second, Ruder [66] also recommend optimizers that internally use adaptive learning rates, like Adam, in the case of training complex networks where we care about fast convergence, as it converges faster than standard SGD [66]. As we will be training many networks during both phases of our optimization, this should decrease the required time needed to find a good model.

Adam introduces 3 hyperparameters besides the learning rate. β_1 and β_2 are

related to how fast the influence of the moving average over the gradient of the previous steps on the update rule will diminish, with higher values leading to a slower decay. Epsilon is set to a small value in order to prevent division by 0 in the update rule [43].
The number of epochs is allowed to vary between 50 and 200. In early experiments even the most complex 5 layer networks converged before 120 epochs with the default settings on all images. Increasing the upper bound to 200 should allow for a proper evaluation of configurations that take longer to converge (e.g. with a lower learning rate) while still keeping computation time manageable. Powers of 2 are often tested for the batch size in a range such as $[32, 512]$, which is the range we will use, where typically lower values lead to an increase in testing accuracy, but also an increase in training time required [41]. The epsilon parameter has a default value of 10^{-7}, however in the keras documentation it is mentioned for certain tasks a good value might be as high as 1 [2]. We trace it in the logarithmic range $[10^{-8}, 1]$. We will copy the ranges used for β_1 and β_2 in one of the experiments in the original paper, $[0, 0.9]$ and $[0.99, 0.9999]$ respectively [43]. Finally we also use the range for the learning rate in that same experiment, which has the logarithmic range $[10^{-5}, 10^{-1}]$.
As we are dealing with multi-class classification we use the categorical crossentropy loss function during training. Training data is scaled to fall in the range $[-1, 1]$ for numerical stability, and the test set is scaled using the same factors [36]. For our hidden layer activation function we use the Rectified Linear Unit function (ReLU). ReLU is faster to compute than activation functions such as tanh, and it is the most common choice in conjunction with softmax in the output layer for most state of the art CNNs [56]. There is no single activation function adopted in the remote sensing literature, with for example Hu et al. [37] using the tanh and Ghamisi et al. [30] using a sigmoid function.

5.7 Support Vector Machine

The first choice to be made when using an SVM is what kernel to use. Depending on the choice, some more hyperparameters are introduced. In the remote sensing literature the Gaussian kernel has shown the best results and is the most commonly used [59, 61, 30]. With this kernel one extra hyperparameter is introduced that controls the size of the kernel, namely gamma. Besides the kernel specific hyperparameters, there is one hyperparameter C present in all SVM variants, which controls the penalty incurred for the misclassification of a datapoint. In Hsu, Chang, Lin, et al. [36] a grid search tracing C in the range $[2^{-5}, 2^{15}]$ and gamma in $[2^{-15}, 2^{3}]$ is recommended, followed by a finer grid search on a promising region. We will use the wide search space for the comparison of our optimization strategies. Furthermore, each attribute should be scaled to avoid numerical problems, which is again done using the training set, to the range $[-1, 1]$ [36].

[2] https://keras.io/api/optimizers/adam/

Table 4: Lower and upper bounds for all models. Bounds are inclusive.

Classifier	Hyperparameter	Lower Bound	Upper Bound
Bagging	Number of trees (nTre)	25	500
RF	Number of trees (nTre)	100	500
RF	Number of features (nFtr)	1	56 (IP/LCVF), 48(UH)
ERT	Number of trees (nTre)	100	500
ERT	Number of features (nFtr)	1	56 (IP/LCVF), 48(UH)
ERT	Minimum samples split (mss)	2	7
SGB	Number of trees (nTre)	500	500
SGB	Learning rate (lr)	0.005	0.15
SGB	Subsample ratio (subs)	0.4	1
SGB	Maximum depth (maxD)	4	8
XGB	Number of trees (nTre)	500	500
XGB	Learning rate (lr)	0.005	0.3
XGB	Row subsample ratio (rSubs)	0.4	1
XGB	Maximum depth (maxD)	1	16
XGB	Column subsample ratio (cSubs)	0.4	1
XGB	Minimum purity (mp)	0	10
XGB	Gamma (gam)	0	1.5
XGB	Lambda (lam)	0	2
CNN (arch)	Number of layers (nLay)	1	5
CNN (arch)	Filter size (filt)	3	25
CNN (arch)	Pooling size (pool)	2 (Layer 1-4), 1 (Layer 5)	5
CNN (arch)	Number of convolutions (nConv)	5	100
CNN (train)	Number of fully connected neurons (nFc)	64	512
CNN (train)	Number of epochs (nEpo)	50	200
CNN (train)	Batch size (bs)	32	512
CNN (train)	Learning rate (lr)	10^{-5}	10^{-1}
CNN (train)	Epsilon (eps)	10^{-8}	1
CNN (train)	Beta 1 (b1)	0	0.9
CNN (train)	Beta 2 (b2)	0.99	0.9999
SVM	C (C)	2^{-5}	2^{15}
SVM	Gamma (gamma)	2^{-15}	2^{3}

5.8 Evaluation

We assess the performance of the seven classification models by using all three optimization strategies when the hyperparameter dimensionality is 3 or lower. For the models with higher dimensional spaces grid search is dropped, for the reasons described in section 3. The experiments ran differ slightly between the datasets.

For our experiments on the IP data we split the labeled samples into 4 mutually exclusive folds. All models are then trained on one of these folds, amounting to 25% of the data, followed by evaluating the results on the remaining 3. As we are directly optimizing for performance on the test data this allows for a direct comparison between optimization strategies, as well as providing an idea of the optimal classifier performance.

The UH dataset has a designated training and testing set, and as a result we run a slightly different experiment compared to IP. In a first step, using 4-fold cross validation on the training data, a promising configuration is found. Note that this time the model is trained on 3 folds and validated on the one held out (the validation set). The model is then trained using these settings on the whole training set, and the final score is obtained by using that model to predict the data in the test set. This experiment gives insight into the generalization capacities of the classifiers, as fully optimizing using cross-validation on the training set does not necessarily imply optimal test set performance (see for example Figure 15).

Due to delayed access to the LCVF data, only BO is run for all models and we directly optimize for test set accuracy using the whole training set, similar to IP.

The accuracy score reported throughout the next sections is the weighted overall accuracy (WOA), calculated by the number of correctly classified samples divided by the total number of samples. In the final comparison, the Kappa coefficient will also be introduced, which corrects for the possibility of similarities occurring by chance. It is defined by

$$\frac{p_0 - p_e}{1 - p_e},$$

where p_0 denotes the agreement between the method and the ground truth, and p_e the expected agreement when the method randomly labels the sample [17]. While the Kappa coefficient has a value between -1 and 1, we multiply it by 100 throughout this work for readability.

6 Results

The results of tuning the models given the ranges as specified in Table 4 are presented here. When comparing different optimization methods the same random seed is used for the building of the models, such that the results differ due

Table 5: Mean and standard deviation over 3 optimization runs of the test set WOA for the optimal bagging models found.

	IP		UH		LCVF	
	Mean	Std	Mean	Std	Mean	Std
Default	76.17%	0.01%	70.62%	0.40%		
Grid	80.44%	0.03%	71.50%	0.09%		
Random	80.44%	0.03%	71.47%	0.11%		
BO	80.45%	0.04%	71.48%	0.09%	82.47%	0.15%

to the quality of the optimization instead of the inherent randomness of the machine learning methods.

6.1 Bagging

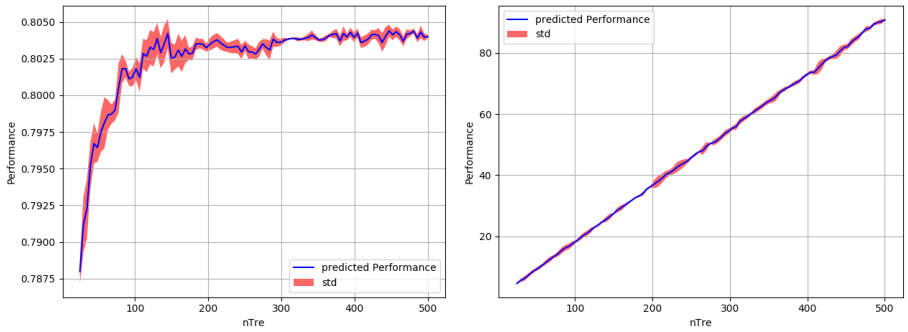
Table 5 shows the mean and standard deviation of the WOA of the best model found over 3 runs for the different optimization methods on the different datasets, each consisting of 25 evaluations. Recall that the number of initial points is set to 10 times the hyperparameter dimensionality and accounts for 40% of the evaluations. In this case we only optimize for the number of trees, hence the total number of evaluations is $10 * 1 * 2.5 = 25$.

As mentioned in subsection 3.4, the method for determining hyperparameter importance can be applied to more measures than the classification performance. In Figure 7 we use the method to assess the effect of the number of trees on both the test WOA and the time required to train the model. After around 200 trees there is a clear plateau in the WOA, but the training time keeps linearly increasing with every added tree. Combining both results shows that using 200 trees likely gives optimal performance and adding more would only increase the time needed to train the model.

6.2 Random Forest

For the grid search we evaluate a structured grid consisting of multiples of 200 for the number of trees and $1+3k$ (k integer) for the number of features up to the upper bound, leading to a total of 57 evaluations on the Indian Pines dataset and 48 for UH. Random search and BO are run for the same number iterations. The results over 3 initializations of the different optimization methods are compared in Table 6.

As RF has only 2 parameters, we can plot the points found together with their test set accuracies and linearly interpolate to get a view of the search space and the approaches the optimization algorithms take. These are shown in Figure 8. The BO has a high concentration of points in the bottom right, where the number of trees is high and the number of features set to around the square root of the input dimensionality. The other parts are explored less as



Test WOA

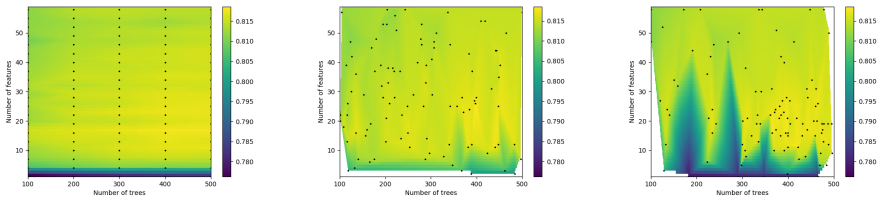
Training time (s)

Figure 7: Effect of the number of trees on the WOA and training time for the IP image.

Table 6: Mean and standard deviation over 3 optimization runs of the test set WOA for the optimal RF models found.

	IP		UH		LCVF	
	Mean	Std	Mean	Std	Mean	Std
Default	81.30%	0.02%	72.88%	0.07%		
Grid	81.85%	0.02%	73.04%	0.13%		
Random	81.85%	0.06%	73.11%	0.02%		
BO	81.88%	0.01%	73.00%	0.07%	85.48%	0.11%

the algorithm does not expect any improvement in those regions, but no section is completely left out. Grid search shows the adapted resolution, with far more values for the number of features tested than for the number of trees. All 3 strategies find similar solutions, and we see the bright yellow regions indicating high performance in the same places.



Grid search Random search Bayesian optimization

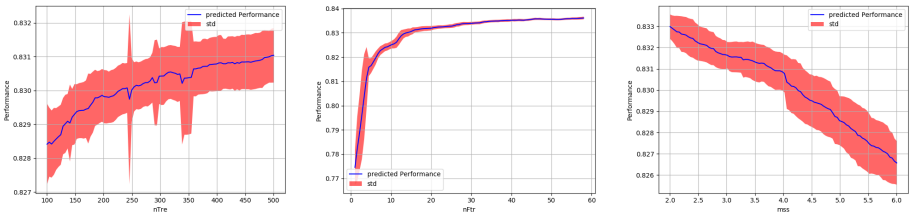
Figure 8: View of the search space obtained by the different optimization methods.

6.3 Extremely Randomized Trees

Using the ranges described in section 5 to run the 3 optimization algorithms, followed by using those results to calculate the importance and the effect of the hyperparameters in the model, we obtain the graphs of Figure 9. As expected, a larger number of trees has a positive effect on the accuracy, and a value of 2 for the minimum samples needed to split a node is optimal. These effects only account for approximately 0.8% and 3.5% of the variance respectively, also reflected by the small range of accuracy values on the y-axes. The number of features however, which is responsible for around 87% of the variability of accuracy, does not show its optimal point as expected at the square root of the input dimensionality. Instead, it seems that an even higher value than was initially included in the search space might result in better performance.

Running BO for 200 iterations, while allowing the number of features to range from the minimal value of 1 to the maximal possible value of the number of bands, and again plotting its marginal accuracy results in Figure 10. Here we see that the initial upper bound was more or less the maximum of the marginal, and BO favors a value of around 60 for the number of features. We re-run the experiment, now setting no upper bound for number of features which gives the results of Table 7 when each optimization is run 3 times for 90 iterations.

As mentioned in subsection 3.4 the results of the fANOVA depend heavily on the chosen hyperparameter ranges. To illustrate this, if instead of using the original ranges we were to run the ERT optimization with the number of trees varying between 1 and 100 and the number of features between 40 and 80, the importance of the number of features drops from 87% to 5% while the



Number of trees | Number of features | Minimum samples split

Figure 9: Effect of individual hyperparameters for ERT on the WOA for the IP dataset. Note the differences in y-axes range, a reflection of relative hyperparameter importance.

Table 7: Mean and standard deviation over 3 optimization runs of the test set WOA for the optimal ERT models found.

	IP		UH		LCVF	
	Mean	Std	Mean	Std	Mean	Std
Default	82.93%	0.02%	73.75%	0.05%		
Grid	83.94%	0.03%	73.53%	0.12%		
Random	83.90%	0.01%	73.62%	0.24%		
BO	83.96%	0.02%	73.65%	0.09%	87.26%	0.26%

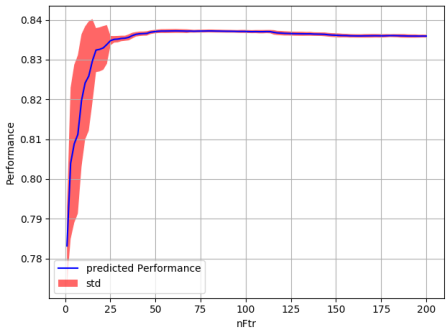


Figure 10: Effect on the WOA for all possible values of the number of features

importance of the number of trees increases to 63%. Hence, the fANOVA results only give an indication about the relative importance on the experiment that was ran, and should not be taken as a general truth about the classifier.

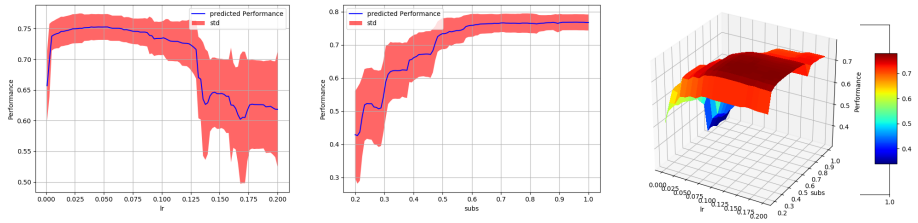
Table 8: Mean and standard deviation over 3 optimization runs of the test set WOA for the optimal SGB models found.

	IP		UH		LCVF	
	Mean	Std	Mean	Std	Mean	Std
Default	79.37%	0.12%	68.27%	0.11%		
Random	81.84%	0.06%	72.97%	0.24%		
BO	81.83%	0.06%	72.87%	0.22%	76.10%	0.37%

6.4 Stochastic Gradient Boosting

Where we saw one hyperparameter dominate the variance in test set accuracy for ERT, this is not the case for SGB. No single hyperparameter or interaction between hyperparameters accounts for over a third of the variance on either the IP or the UH dataset. The subsample ratio comes the closest, responsible for around a quarter. Furthermore the individual hyperparameters together represent only about half of the variance in WOA given the ranges specified in Table 4.

Take for example the three most influential effects on the test set WOA, depicted in Figure 11. The empirical recommendation of setting the learning rate to below 0.1 shows up, and a subsample ratio of over 0.6 seems optimal. The interaction between them however favors a higher learning rate with a lower subsample ratio. The optimal model settings will thus probably be somewhere where the high values for these 3 graphs overlap, but again not even half of the variance is accounted for. The accuracies over 3 instances of 100 iterations can be found in Table 8.



Learning rate Subsample ratio Interaction

Figure 11: Effect of the subsample and learning rate as well as their interaction on the test set accuracy for the IP dataset.

Table 9: Mean and standard deviation over 3 optimization runs of the test set WOA for the optimal XGB models found.

	IP		UH		LCVF	
	Mean	Std	Mean	Std	Mean	Std
Default	82.66%	0.00%	73.07%	0.00%		
Random	84.05%	0.05%	73.74%	0.17%		
BO	84.44%	0.13%	73.82%	0.28%	83.66%	0.77%

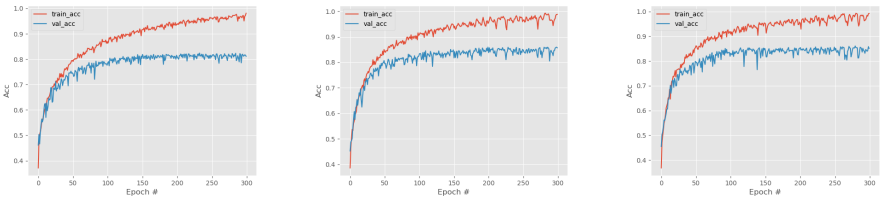
6.5 Extreme Gradient Boosting

As we are optimizing for 8 hyperparameters, we evaluate each optimization strategy three times each for 200 iterations, after which the scores given in Table 9 are found. Both by looking at the marginals as well as the tendencies of the Bayesian optimization algorithm it seems that the ranges chosen for the hyperparameters are not optimal. For example, the highest scoring model on the UH dataset uses the lower bound for the row and column subsample ratio, gamma and the minimum purity as well as the upper bound for the learning rate. Only in two of these cases, for the minimum purity and gamma, both set to 0, this bound cannot be extended further in that direction. Hence, either the ranges of the already comparatively large 8-dimensional search space will have to be expanded even further, which will in turn require more iterations to find an optimum, or we can shift the ranges towards where we expect an optimum to be, which introduces a bias.
The first approach did not result in finding models with higher WOA. Using the second approach however, we can increase the test set accuracy for IP to 85.61, over 1 percentage point higher than the best models found using the first bounds. On the UH data this approach is less successful, and the best model found remains the one with many hyperparameters on the edge of the search space. The highest values found will be used for the comparison in the next chapter.

6.6 Convolutional Neural Networks

The first heuristic, combining Saxe et al. [68] and Hahn et al. [32] by classifying completely random weights with a fully connected layer for 30 epochs, does not seem to be a good indication of the quality of the complete architecture. Optimizing on the Indian Pines image gives average accuracy scores for different architectures ranging from around 60% to as low as 25%. When fully training three of these networks with a large difference in random weight scores the ordering is not preserved. Figure 12 shows this, where the top scoring model on the heuristic scores lowest when fully trained, while the other 2 give comparable results despite a difference of over 10% on the random weight classification. In

general, the more layers the model has, the lower the random weight score is using this approach.

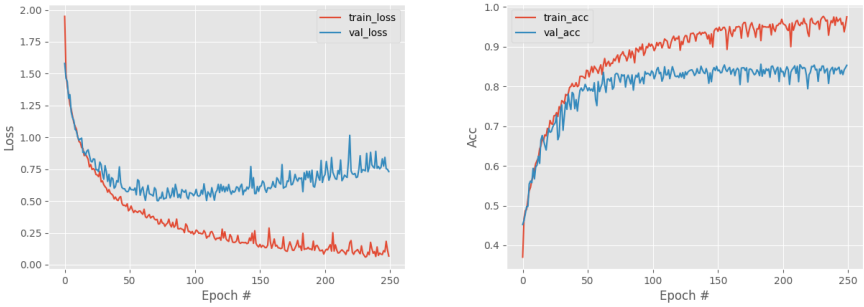


61.38% 48.83% 37.37%

Figure 12: Train and test accuracies after fully training architectures that scored the percentage given in the subcaption for the heuristic.

When following Hahn et al. [32] entirely and training the convolutional layers for 3 epochs followed by the classification layer for 30 epochs the best performing networks found by the Bayesian optimization all consist of 2 layers, and the clear ordering of models with less layers outperforming the more complex ones is no longer present, although the bottom half is mostly comprised of the larger models with 4 or 5 layers. With the highest test accuracy found at 77.37% and all but one of the 50 models scoring below the previously found maximum of around 60%, training the feature extraction part for only 3 epochs has a surprisingly large impact on the test accuracies found. The results are also getting closer to the findings in Hahn et al. [32], where the heuristic reaches accuracies almost one percentage point below that of a fully trained model. The highest scoring model for example scores only around 3 percentage points lower using the heuristic than when fully training the model.
A 5-layer model scoring 70.55% with this heuristic scores higher than the model with the highest heuristic score (77.37%) after 200 epochs, with around a 83% test accuracy compared to ±80.5%. A 4-layer model reaching 61.66% accuracy in the heuristic achieves similar scores (±83%) after full training.
With the third heuristic, running a BO loop when using a linear SVM with a small grid search to determine random weight performance, we again run into the same problem. All the highest scoring models are single layer models that end up performing worse than the deeper networks upon full training. It is nonetheless interesting that classifying completely random weights, while only checking 3 values for C in the linear SVM, the test WOA can reach up to 80% on the IP data.
Clearly the current heuristics do not allow for the more complex models to compete with the simpler ones, even though they can outperform them when fully trained. One solution would be to increase the number of epochs we train either the convolutional layers, the fully connected layers or both. The most complex model in our search space has to be able to reach a score close to its potential. Alternatively we can use models of similar complexity and stick to the heuristic.

We will use the former.
Assuming that models with more layers will take longer to converge, we looked into the convergence of some of these deep networks. After fully training for around 110 epochs, the test accuracy had plateaued for all 5-layer models tested on the IP image. This happened at around 70 epochs on UH. We will use these as the number of epochs all layers in the model are trained for during the optimization. Clearly this does not resemble the original heuristics anymore, and it is a time-consuming process. Note that we do not need to worry about invalid architectures and adapt the search space when switching between datasets, instead by returning a score of 0 for invalid models the BO will learn to avoid these architectures, and random search will simply continue with the next architecture.
The loss and accuracy graphs for all models tested follow the pattern seen in Figure 13. There is a clear sign of overfitting where the training and test loss diverge after some number of epochs, but both training and testing accuracy do not decrease. For our heuristic this allows us to increase the number of epochs trained without the worry of overfitting the simpler models and misrepresenting their potential.
What makes these comparisons more difficult are the per-epoch fluctuations



Loss Accuracy

Figure 13: Typical training and testing loss and accuracy graphs. Note that even though there are clear signs of overfitting after 100 epochs where the losses start to diverge, the testing accuracy does not decrease.

in both the training and testing accuracy and loss, similar to what is seen in Figure 12. As these fluctuations do not appear in the same places for the different folds, this makes it difficult to set the number of epochs to a value that consistently gives a high accuracy. Taking again the highest scoring model as an example, there is a difference of around 5 percentage points in test accuracy between the 4 folds after 200 epochs. The maximum testing accuracies obtained during the epochs on the other hand only differ by 1,5 percentage point between the folds. Slightly decreasing the learning rate does not solve the issue. While

Table 10: Mean and standard deviation of test WOA in percentages with CNN, using different strategies for picking the number of epochs. The Full column give the scores of the model after all epochs, Loss gives the performance at the point of minimum training loss and Acc when the training accuracy is at its highest.

	IP			UH		
Layers	Full	Loss	Acc	Full	Loss	Acc
1	79.47 ± 1.88	80.79 ± 0.64	80.60 ± 0.39	74.80 ± 0.55	74.20 ± 0.63	75.32 ± 1.09
2	79.78 ± 2.53	81.56 ± 0.79	81.49 ± 0.68	74.87 ± 1.00	74.40 ± 1.66	74.40 ± 1.56
3	81.74 ± 0.78	82.01 ± 0.61	82.32 ± 0.82	75.85 ± 0.77	76.24 ± 0.51	76.05 ± 0.36
4	80.24 ± 0.46	81.13 ± 0.68	81.13 ± 0.68	76.08 ± 0.25	75.57 ± 0.33	75.63 ± 0.31
5	79.55 ± 1.06	80.73 ± 0.75	80.73 ± 0.75	76.37 ± 0.36	76.61 ± 1.00	76.39 ± 1.02

lowering the learning rate further should result in a smooth curve, this will greatly lengthen the time needed to form an idea of the potential of a candidate architecture, and as a result hamper the optimization procedure.
Instead, as visually there does seem to be some overlap in the peaks of the training accuracy with that of the test accuracy, we investigate if we can use either this or the training loss to determine for how many epochs to train our model.
In Table 11 the results are given of training different sized architectures on IP and UH, using three different strategies for determining how many epochs to train for: the model after all epochs, the point where training loss was minimal and the point of maximal training accuracy. For the IP data the minimal loss or maximum accuracy give overall better results, while for UH they are similar. From here on out we use the maximum training accuracy as the determining factor for choosing the number of epochs.

With all of the above in mind, we use TPE-based BO to optimize over the defined search space on each image 3 times for 50 iterations, which was found to still be computationally manageable. The best models were then trained for 500 epochs and the mean and standard deviation are presented in Table 11. Since we have access to the test set beforehand, besides the score at the epoch with the highest training accuracy we also report the peak test WOA in the table. This can give an indication of what the performance could be after optimal training, but is not used in the comparisons. The BO and random search made use of a different batch size during optimizing, hence a part of the difference between them may be attributable to this.

The best architecture found for each model is optimized in a second BO loop for 150 iterations. We limited the number of epochs to 200 while optimizing as the performance always plateaued before that point during initial testing and to keep computation time manageable. However, after more training the model keeps very slowly increasing its WOA. As a result, the highest score during the BO loop did not always result in the best final model. On the IP dataset the best configuration reached 82.89% test accuracy after 170 epochs, and 92.97%

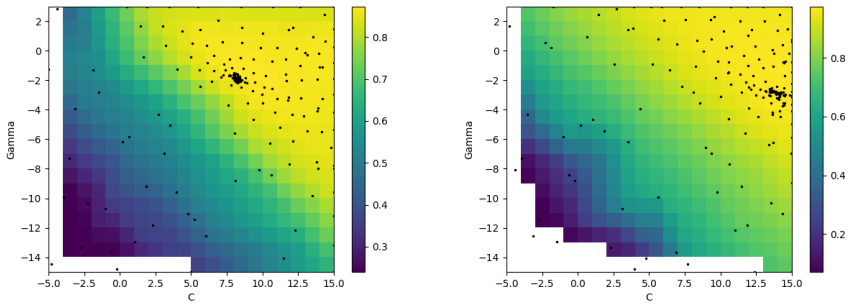
Table 11: Mean and standard deviation over 3 optimization runs of the test set WOA for the optimal CNN models found.

	IP		UH		LCVF	
Random (acc)	85.50%	1.03%	77.51%	0.39%		
Random (max)	86.11%	0.99%	78.73%	0.29%		
BO (acc)	84.68%	0.66%	77.20%	0.62%	89.06%	0.59%
BO (max)	85.30%	0.65%	78.75%	0.42%	89.79%	0.24%

training WOA. Full training then resulted in a 83.96% test WOA. A configuration with slightly lower test accuracy but much lower training accuracy in 167 epochs, 81.92% and 84.68% respectively, reached 86.49% test WOA after full training. This is unlike the UH image, where the model with 95.41% validation and 96.70% training WOA achieved 0.11% higher on the test set compared to the model with 94.77% validation and 90.67% training WOA. The highest scoring settings after full training are used in the comparison in section 7.

6.7 Support Vector Machine

The results of running the SVM for 90 iterations are given in Table 12. The number of iterations was chosen to follow the grid search of Hsu, Chang, Lin, et al. [36]. As we are again dealing with a 2-dimensional search space we can visualize the tendencies of the BO algorithm, shown in Figure 14. The optimal configuration for UH seems to be on the edge of the defined search space, and perhaps lies outside of it. To test this we ran another BO loop where the upper bound for gamma and C were extended, but the resulting optimal configuration did not improve upon the one previously found.



IP UH

Figure 14: SVM search spaces using BO on 2 different datasets. Note that the axes are on a binary logarithmic scale.

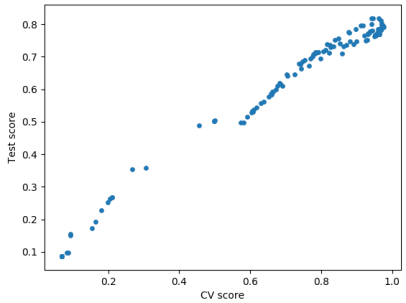


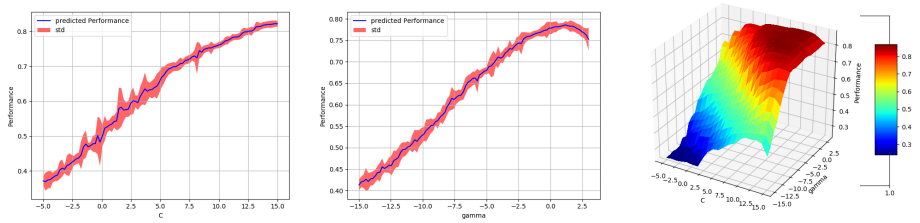
Figure 15: Cross validation versus testing accuracy for SVM.

Scaling the training data from its original values to $[-1, 1]$ and the test data by the same factor increases the performance by a factor 10 approximately. As this has such a huge effect on the accuracy it would be interesting to consider this as another aspect to tune. Consequently, we add 2 hyperparameters to our 2-dimensional search space. The first one, x, scales the data to the interval $[0, x]$, with the second one, y, acting as the offset. This then results in the training data being scaled to the interval $[-y, x - y]$.
Running BO for 200 iterations does not improve the score found during the optimization. Similar to ERT however, where the default values slightly outperform the optimized values on the UH test set, even though the default values result in a lower score during cross validation, the best SVM configuration found in this 4-dimensional space does in fact reach a higher accuracy and kappa score on the UH test set while having a lower cross validation score.
This raises the question whether the training set is a good representation of the larger test set, and if fully optimizing on the training set is the most effective strategy. In Figure 15 we plot the accuracy found during the optimization with cross validation against the testing accuracy of the same model using the original 2-dimensional hyperparameter space. The expected diagonal line largely shows up, but the optimal model on the testing data is in fact not the optimal configuration found during cross validation on the training set. The maximal testing accuracy is 81.87%, which reached a cross validation score of 94.74% (C = 11094.06 and gamma = 0.02), compared to the 79.17% test WOA of the model with the highest cross validation score of 97.57% (C = 11587.60 and gamma = 0.15). As there is no clear way to determine from the training set alone what model will score highest on the testing data, we stick with the optimal model found during the first phase.

According to the fANOVA, the C parameter is the most important, accounting for around 52% of the variance on IP. It is followed by gamma which represents ±39%, and their interaction the remaining 9%. Plots of their effects are given in Figure 16, where we see that high C values and gamma values slightly below the upper bound are considered optimal.

Table 12: Mean and standard deviation over 3 optimization runs of the test set WOA for the optimal SVM models found.

	IP		UH		LCVF	
	Mean	Std	Mean	Std	Mean	Std
Default	51.85%	0.00%	66.34%	0.00%		
Grid	86.96%	0.00%	79.37%	0.00%		
Random	86.95%	0.20%	78.99%	0.20%		
BO	87.26%	0.02%	79.34%	0.10%	86.49%	0.01%



C

gamma

Interaction between C and gamma

Figure 16: Effect of the 2 SVM hyperparameters and their interaction on the WOA on the IP image. Note that the axes are on a binary logarithmic scale.

7 Comparison

This section starts with a comparison between the seven classifiers, followed by an evaluation of the hyperparameter tuning strategies.

7.1 Models

A comparison of the best scoring models is given in Table 13, alongside a comparison between fit and predict times in Table 14. These run times can vary heavily between similar scoring models on the same image and as such should be taken with a grain of salt. Consider for example the highest scoring RF model on the UH dataset, which constructs 160 trees using 14 features. The second best model, only scoring around 0.001% WOA lower on the test set, builds 486 trees with the same number of features. As a result this second model takes over triple the time to fit compared to the first model.
For the IP data the mean and standard deviation are calculated over the 4 folds, using 1 fold for training and the remaining 3 for testing. Since there is a designated training and testing set for both the UH and LCVF data, for these images the values are computed by training the optimal model on the whole training set and evaluating on the test set 4 times with different random seeds.

Out of bagging, RF and ERT there is a clear ordering in quality when ap-

Table 13: Mean and standard deviation of the accuracy and kappa score for the optimal models found.

	IP				UH				LCVF			
	Acc		κ		Acc		κ		Acc		κ	
	Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std
Bagging	80.49%	0.42%	77.56	0.47	71.59%	0.06%	69.50	0.06	82.21%	0.35%	81.25	0.46
RF	81.91%	0.24%	79.19	0.27	73.09%	0.03%	71.09	0.03	85.34%	0.18%	84.56	0.23
ERT	83.98%	0.23%	81.61	0.26	73.79%	0.13%	71.83	0.15	86.99%	0.47%	86.28	0.60
SGB	81.91%	0.19%	79.20	0.21	73.07%	0.33%	71.04	0.36	75.72%	0.39%	74.43	0.40
XGB	85.64%	0.11%	83.52	0.11	73.93%	0.17%	71.81	0.22	84.49%	0.43%	83.65	0.45
CNN	86.49%	0.81%	84.57	0.96	77.49%	0.50%	75.63	0.54	88.80%	0.63%	88.19	0.67
SVM	87.33%	0.15%	85.53	0.15	79.17%	0.00%	77.58	0.00	86.43%	0.00%	85.68	0.00

Table 14: Mean and standard deviation of the fit and predict time in CPU seconds for the optimal models found.

	IP				UH				LCVF			
	fit		predict		fit		predict		fit		predict	
	Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std
Bagging	23.21	0.39	0.13	0.00	58.26	0.05	0.51	0.00	5.21	0.02	0.09	0.00
RF	6.37	0.04	0.35	0.00	2.80	0.02	0.22	0.00	0.34	0.00	0.20	0.00
ERT	5.67	0.99	0.58	0.00	0.36	0.02	0.20	0.00	0.40	0.00	0.19	0.00
SGB	314.19	5.61	0.60	0.03	73.46	0.55	1.94	0.01	27.58	0.22	0.87	0.01
XGB	20.39	0.07	3.46	0.02	51.56	0.13	5.67	0.02	11.30	0.01	1.53	0.03
CNN	5172.50	1184.78	4.44	0.05	3518.78	583.95	5.03	0.11	174.28	13.40	0.89	0.02
SVM	0.52	0.01	2.27	0.06	0.26	0.00	1.21	0.00	0.07	0.00	0.45	0.00

plied to the datasets used in this work. Bagging performs worst while also being the slowest of the three, and even though RF and ERT both only need a few seconds to train, ERT outperforms RF in both WOA and kappa score. Where RF is usually seen as a good model to get a baseline on a hyperspectral data classification problem that does not much optimization, this can instead be done with ERT. Similar to RF the performance is mostly dependant on the number of features considered, but using the square root of the input dimensionality while setting the number of trees to a high enough number and keeping the number of samples required to split a node at 2 consistently produces good results. Even though the number of features favored by the optimization can be much higher than the recommendation, as shown in Table 15 and Figure 10, the fact that the default settings actually outperform the optimized variants on the UH data (Table 7) shows its robustness.

As for the 2 gradient boosting models, there does not seem to be any reason for using SGB. It achieves similar performance to RF, while having more hyperparameters that are harder to tune and taking longer to train. XGB is a more nuanced case. It takes a similar amount of time to train given the optimal settings found as bagging, and achieves the highest performance out of the 5 tree ensemble classifiers considered. Tuning the 8 hyperparameters however is a difficult problem. Given the fact that the optimal configurations found on all 3 images include multiple parameters on the edges of their defined ranges, it is possible the performances reported here may be improved upon.

It should be noted that we ran into some issues while running parallel instances of XGB, as they attempt to use multiple cores each. This was solved by restricting them to a single core, and as such the times seen in Table 14 are obtained by using a single core. The models will likely be faster when ran in isolation.

The SVM, as has been established in many articles before, is a good choice for classifying hyperspectral imagery. Not only is it the fastest method to fit to the data, it also achieves the highest scores on two of our datasets. Furthermore, the speed and low hyperparameter dimensionality make it relatively easy to tune the model, as many configurations can be tried in a short period of time, and as can be seen in Figure 14 the location of the maximum within the search space spans a wide enough area to be found by a coarse grid search or a few random search iterations.

While the CNN scores are only slightly lower than the SVM on IP and UH and slightly higher on LCVF, the tuning process is the exact opposite, being both time consuming and unintuitive.

The dimensionality and possible size of the CNN search space is huge. For example, many other pooling layers such as average pooling can be used, the order of layers could be switched up by having multiple convolutional layers before the pooling operation or by using more than two fully connected layers as the classifier, we can allow for more than 100 convolutions, etc. As a result, even though many architectures have been tested, there is no guarantee that the results found in this work will be optimal.

The large standard deviations for the CNN fit times are a result of early stopping, where the training process is halted once the training accuracy plateaus

before the set number of epochs. The architectures found were able to reach 100% training accuracy on all 3 images. All optimal configurations for the tree ensembles also reached 100% training accuracy, hence this metric is not included in the comparison tables.
From Table 13 we see that there is a larger gap in performance between the tree ensembles and CNN/SVM on UH compared to IP, as well as larger differences amongst the tree ensembles on IP compared to UH. Classifiers such as XGB perform relatively better when optimized directly for test set performance, than when used to generalize to the test set from an optimal model found on the training set. An explanation is that they are overfitting, learning characteristics that are too specific and do not translate well to a slightly different test set. Table 13 also shows that the ordering in quality of the classifiers is different for the LCVF image when compared to both UH and IP. The performance of SGB dropped drastically and is now surpassed by bagging, and both ERT and the CNN outperform the SVM. A possible explanation for these results is the fact that the LCVF image has more classes (23 compared to 15 and 16), while also having less training samples, and some classes containing as little as 7 spectra. In contrast, the class with the lowest number of samples in the UH image has 181 training spectra, while the largest LCVF class has size 160, and 2832 total training samples compared to 931. The IP image does contain classes with as little as 5 or 7 training samples, but is the most imbalanced, including classes up to a size of 614 for a total of 2562 training samples. Another factor could be that, unlike for UH and IP, all spectra of the LCVF image are selected manually. As a result, they might be a better characterization of their classes.

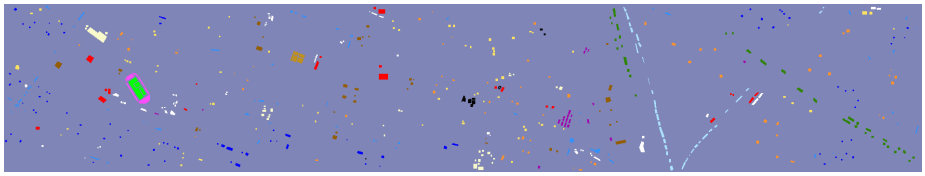
We provide a visual comparison on the UH image in Figure 17. The optimal classifiers trained on the whole training set are used to predict all spectra in the image. For classes that achieve similar scores, e.g. RF and SGB or ERT and XGB there are still obvious differences between their predictions on the whole image. Consider for example the large region on the right where XGB predicts mostly trees (yellow), whereas ERT finds water (black) and residential areas (blue). While this is the most striking example, differences appear throughout the images, mostly at unlabeled pixels that likely do not belong to one of the 15 classes. In Figure 18 we show a typical example of the certainty with which the pixels are classified. In the large region to the right all pixels are classified with a low probability, leading to a less meaningful comparison in these areas. One could instead set a threshold, such that pixels that are classified with low certainty will remain unlabeled. This can lead to interesting discoveries. Unlabeled pixels within groups of similarly labeled pixels could be a new class, and unlabeled pixels between two classes could indicate they were not fully represented by their training samples [52].

After the training of tree based models we can get an idea of what bands were most important during the training of the model. This is done for each band by calculating the average purity gain over all splits that were made when using a specific band. The higher this metric - called the impurity importance - is, the more important the feature is for discriminating between spectra using said classifier [11].

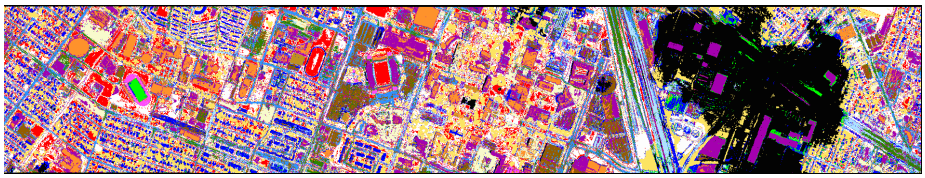
Table 15: Optimal configurations for each model on the different datasets.

Bagging	nTre									
IP	126									
UH	363									
LCVF	159									
RF	nTre	nFtr								
IP	329	20								
UH	160	14								
LCVF	327	1								
ERT	nTre	nFtr	mss							
IP	491	58	2							
UH	100	12	2							
LCVF	328	15	2							
SGB	nTre	subs	lr	maxD						
IP	500	0.90	0.08	4						
UH	474	0.44	0.06	6						
LCVF	500	0.62	0.30	4						
XGB	nTre	rSubs	lr	maxD	cSubs	mp	gam	lam		
IP	500	0.75	0.19	4	0.1	0.06	0.00	1.00		
UH	500	0.40	0.15	7	0.4	0.00	0.00	1.00		
LCVF	500	0.40	0.30	2	0.40	0.00	0.00	0.00		
CNN (arch)	conv1	filt1	max1	conv2	filt2	max2	conv3	filt3	max3	fc1
IP	57	4	3	51	6	4	88	4	3	151
UH	67	5	4	41	5	2	71	5	3	498
LCVF	14	9	2	-	-	-	-	-	-	155
CNN (train)	lr	bs	eps	b1	b2					
IP	$10^{-2.88}$	68	$10^{-2.32}$	0.69	0.99					
UH	$10^{-2.33}$	32	$10^{-6.07}$	0.40	0.99					
LCVF	$10^{-2.57}$	52	$10^{-1.93}$	0.49	0.99					
SVM	C	gamma								
IP	$2^{8.22}$	0.27								
UH	$2^{13.50}$	0.15								
LCVF	$2^{9.83}$	0.002								

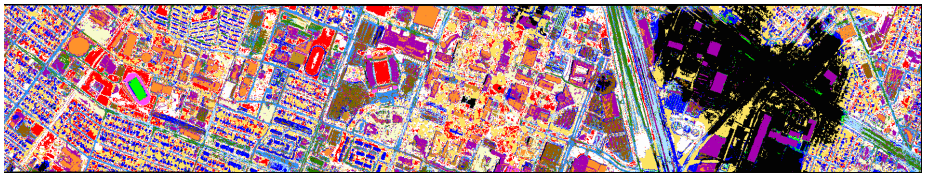
Color composite of the UH image, with band 70 as R, 50 as G and 20 as B [30].



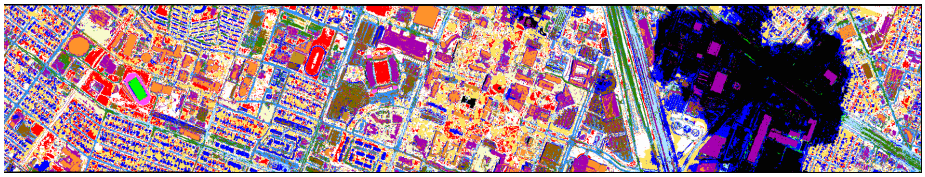
All labeled samples



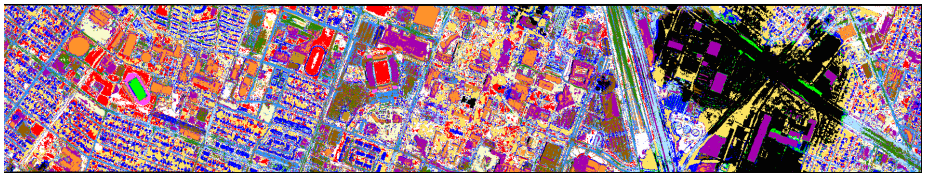
Bagging



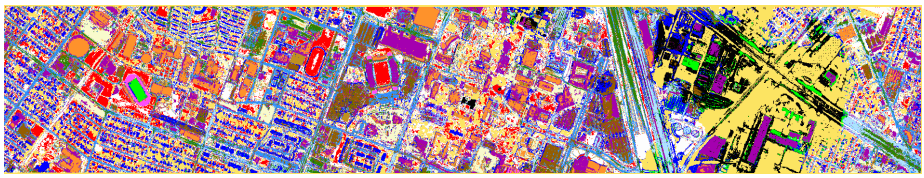
Random Forest



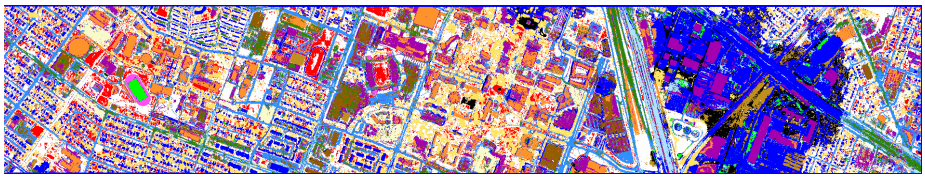
Extremely Randomized Trees



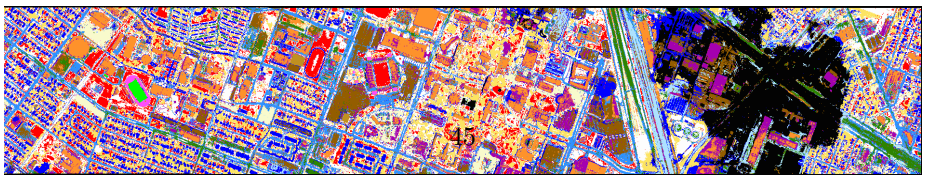
Stochastic Gradient Boosting



Extreme Gradient Boosting



Convolutional Neural Network



Support Vector Machine

Legend, see Table 2 for class names

Figure 17: Classification maps of the optimal configurations on UH.

Figure 18: The certainty with which the SVM classified each pixel on the UH image. Black means 0%, while white represents 100% certainty.

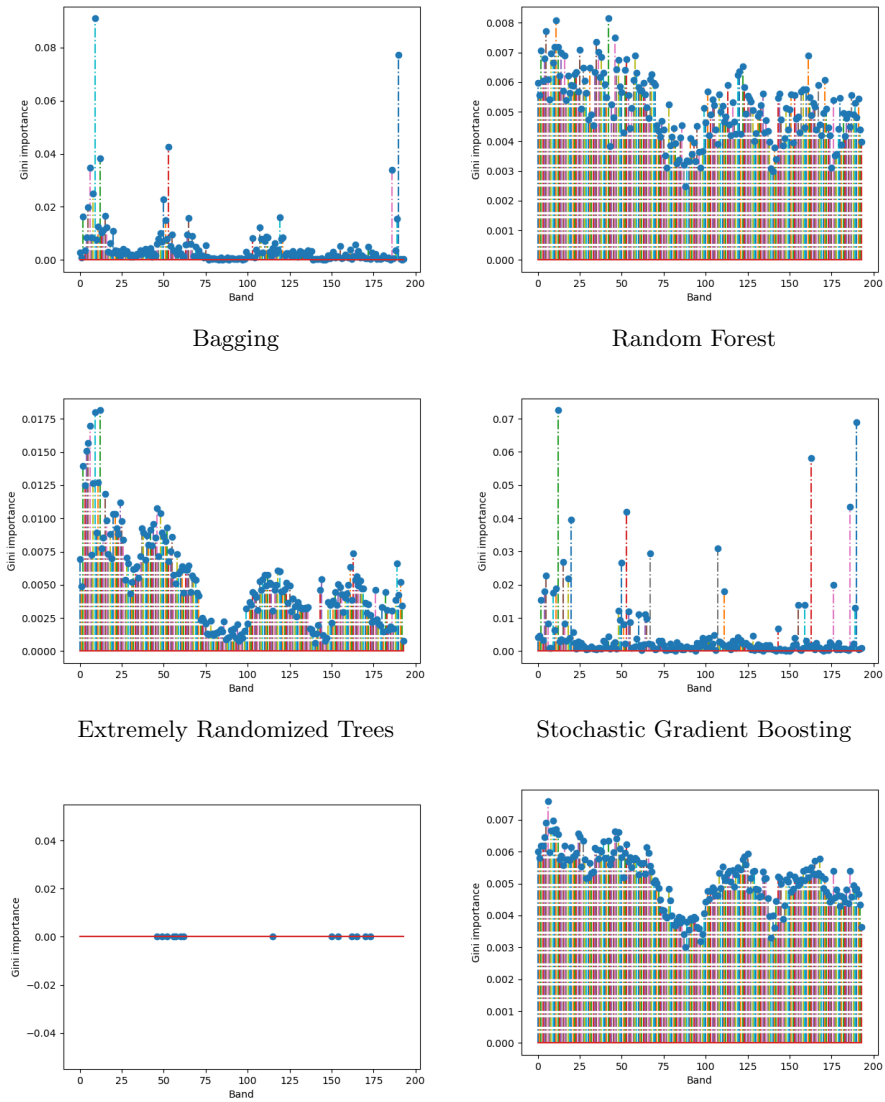
We calculate the impurity importance for all 5 tree ensembles on the LCVF dataset and normalize the results to sum to 1. The results are shown in Figure 19. Both bagging and SGB have access to all features at every split, which is reflected in a few very high peaks in the variable importance. ERT and RF on the other hand subsample the features at each node and as a result have a much more evenly spread out distribution. This is especially the case for RF as the optimal configuration found on LCVF only used a single randomly selected feature at every split. Nonetheless, we see similar patterns for both models, where the bands in the range between 75 and 100 are the least useful for the classification and peaks emerge around bands 10, 50, 120 and 165 (after preprocessing). The most important bands with their importance can be found in Table 16.

Due to the peculiar settings found for XGB by the optimization, with among other things a maximum depth of only 2, only 14 bands are ever used to split on. As a result, many variables have an undefined importance. This is likely also the reason it gives a variable importance of 0 for the bands that are actually being used. When running XGB with different configurations, such as with the default settings, all features are used and the importance again sums to 1. Hence, this is unlikely due to an error in the implementation.

Probst and Boulesteix [62] suggest that variable importance measure estimates become more precise when using more trees. We try this for our RF by building ten times the original number of trees. As we see in the final panel of Figure 19 indeed the distribution is smoother than original RF case. This model with 3270 trees scored 0.3 percentage point less WOA on the test set compared to the original model with 327 trees.

We can compare our importance measures with an analysis done in Mendenhall and Merényi [51], where they identify the most important features on the LCVF data using an improved version of generalized relevance learning vector quantization (GRLVQ). Their results look similar to the ERT and RF results, with 2 peaks followed by an unimportant section and another peak. There is however one key difference: the last peak, around band 165, is completely absent. Why this is warrants further research, but one contributing factor might be that the work of Mendenhall and Merényi [51] is done only with the training set of the LCVF data.

In Table 17 we give the user and producer accuracy (the remote sensing equivalents of precision and recall [76]) on the LCVF image to see whether the



Bagging

Random Forest

Extremely Randomized Trees

Stochastic Gradient Boosting

Extreme Gradient Boosting

Random Forest with 10 times the number of trees

Figure 19: Variable importance for the optimal LCVF models.

Table 16: The 5 most important bands for each tree ensemble classifier by normalized purity gain.

Classifier	Band	Importance
BG	6	0.035
BG	12	0.038
BG	53	0.043
BG	190	0.077
BG	9	0.090
RF	35	0.007
RF	46	0.007
RF	5	0.008
RF	11	0.008
RF	42	0.008
ERT	4	0.015
ERT	5	0.016
ERT	6	0.017
ERT	9	0.018
ERT	12	0.018
SGB	53	0.042
SGB	186	0.044
SGB	163	0.058
SGB	190	0.069
SGB	12	0.072
XGB	71	0.000
XGB	69	0.000
XGB	67	0.000
XGB	68	0.000
XGB	193	0.000

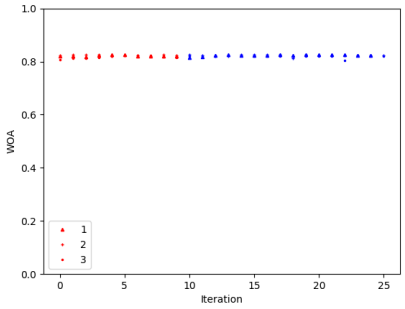
small differences in training sets between Merényi et al. [52] and this work, as described in section 4, have an impact on the performance and thus the comparison. Class G, J and V have 14 training samples in Merényi et al. [52]. In this work they have 7, 12 and 12 respectively. If the differences in accuracies between the studies for one of these three classes is noticeably different than the differences in accuracies between the studies for the other 20 classes, we will have to disregard that class in our comparison.
We add the results of the best model in Merényi et al. [52], the SOM-hybrid ANN, to Table 17. The SOM-hybrid ANN reached a WOA of 88.71% with a Kappa score of 88.11. We compare it with its closest competitors: CNN, SVM and ERT. Note that the CNN scores extremely similar, with a WOA of 88.80% and a Kappa score of 88.19. Only in one case, the user accuracy when compared to ERT, does one of these classes (V) have the largest difference in accuracy out of all 23 classes. The performance differences when classifying V are generally the largest out of the three, but differences for unchanged classes such as S are typically even larger. Hence, there does not seem to be an indication that the slightly modified training set had a large impact on the results. While this does not mean that the comparison is completely accurate, it should still provide a good idea of the relative quality of the classifiers on the LCVF image.

7.2 Optimization Strategies

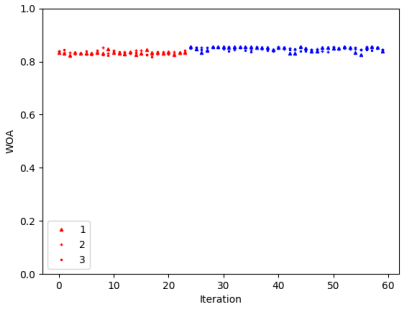
Overall the models under consideration do not benefit from a more sophisticated optimization algorithm than grid or random search, given the number of iterations they were used for. The two minor exceptions on the IP data to this rule are SVM and XGB, where the BO algorithm finds configurations that score higher than the other strategies by over 1 standard deviation. On the UH data this does not happen. This is reflected in the convergence of the BO, as seen in Figure 20, where during the initial phase of the optimization loop a configuration close to the optimal one is already found for every model except XGB.
The three CNN architecture search heuristics considered were not successful in predicting the performance of the fully trained models. All three favored shallow networks which were outperformed upon full training on UH and IP. As the best LCVF model found consists of a single layer, the heuristics might have been successful on that data, but due to the delayed access to the image we were unable to investigate this further. Furthermore, even though the performance of the CNNs during training plateaued well before 200 epochs, the slow but steady rise that followed led to the optimization loop being unable to find the best model for IP when using 200 as the upper bound for the number of epochs. All in all, the best choice might have been to consider all CNN hyperparameters at the same time while optimizing until the performance truly plateaued, even though this would have been far more time-consuming. An alternative can be found in the very recent Chen et al. [15], in which automatic CNN design was applied to remote sensing for the first time using a gradient descent based search, where they were able to achieve competitive results on 4 hyperspectral datasets.

Table 17: User (U) and producer (P) accuracy for all optimal models on LCVF as well as the SOM-hybrid ANN from Merényi et al. [52].

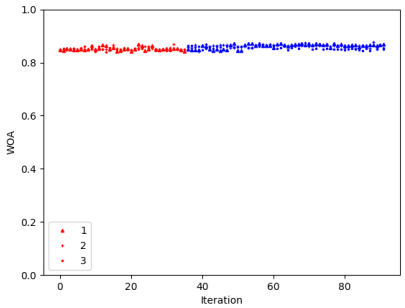
Class	Bagging		RF		ERT		SGB		XGB		CNN		SVM		ANN[52]	
	U	P	U	P	U	P	U	P	U	P	U	P	U	P	U	P
A	0.97	0.99	0.98	0.99	0.98	1	0.98	1	0.98	1	0.98	1	0.98	1	0.98	0.96
B	0.98	0.58	1	0.48	1	0.45	0.9	0.39	0.99	0.45	0.99	0.61	1	0.56	0.95	0.72
C	0.72	0.98	0.83	0.98	0.73	0.98	0.53	0.93	0.65	0.97	0.84	0.93	0.61	0.97	0.95	0.87
D	0.79	0.96	0.88	0.95	0.86	0.96	0.75	0.95	0.86	0.95	0.93	0.95	0.9	0.97	0.92	0.94
E	0.88	1	0.95	1	0.95	1	0.89	1	0.9	1	0.9	0.99	0.9	1	0.94	0.88
F	0.31	1	0.29	1	0.53	1	0.4	1	0.53	1	0.7	1	0.57	1	0.40	0.85
G	1	0.25	1	0.21	1	0.78	0.92	0.77	1	0.77	1	0.78	1	0.88	1.00	0.65
H	0.96	0.93	0.96	0.97	0.95	0.96	0.76	0.84	0.89	0.9	0.95	0.94	0.96	0.94	0.97	1.00
I	0.36	0.87	0.34	0.84	0.64	0.87	0.33	0.68	0.58	0.73	0.7	0.9	0.75	0.86	0.51	0.63
J	1	0.8	1	0.93	1	0.93	0.98	0.76	1	0.92	1	0.98	1	0.91	1.00	0.96
K	0.95	0.71	0.99	0.79	0.99	0.77	0.98	0.78	0.99	0.72	0.98	0.79	0.99	0.79	0.93	0.94
L	0.94	0.99	0.93	1	0.92	1	0.94	1	0.94	1	0.87	1	0.85	1	0.98	0.98
M	0.92	0.66	0.93	0.38	0.94	0.7	0.82	0.24	0.92	0.61	0.88	0.69	0.86	0.27	0.92	0.83
N	0.94	0.73	0.99	0.71	0.99	0.68	0.95	0.49	0.98	0.64	0.98	0.96	0.99	0.89	0.88	0.98
O	0.92	0.91	0.91	0.95	0.91	0.94	0.88	0.8	0.91	0.92	0.92	0.93	0.89	0.94	0.95	0.96
P	0.8	0.95	0.78	0.98	0.78	0.97	0.67	0.95	0.78	0.97	0.86	0.95	0.82	0.95	0.97	0.83
Q	1	0.74	0.98	0.99	1	0.84	1	0.5	1	0.76	1	0.67	1	0.65	0.99	0.96
R	0.55	0.98	1	0.9	0.92	0.98	0.82	0.92	0.88	0.98	0.98	0.98	0.86	0.98	0.88	0.92
S	0.97	0.98	0.98	0.98	0.67	1	0.57	0.93	0.69	0.98	0.5	0.98	0.51	1	0.91	0.80
T	0.9	0.98	0.95	0.98	0.95	0.98	0.55	0.95	0.89	0.98	0.91	0.97	0.95	0.94	0.91	0.92
U	0.64	0.41	0.7	0.77	0.72	0.86	0.45	0.47	0.67	0.86	0.7	0.97	0.66	0.98	0.78	0.74
V	0.41	0.72	0.53	0.7	0.46	0.78	0.42	0.58	0.3	0.68	0.82	0.8	0.84	0.74	1.00	0.62
W	0.85	0.89	0.94	0.9	0.91	0.88	0.89	0.83	0.93	0.89	0.95	0.81	0.96	0.76	0.94	0.92
Avg	0.82	0.83	0.86	0.84	0.86	0.88	0.76	0.77	0.84	0.86	0.88	0.89	0.86	0.87	0.90	0.86



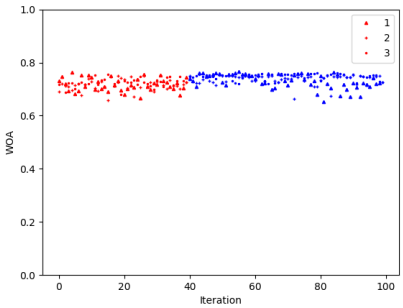
Bagging



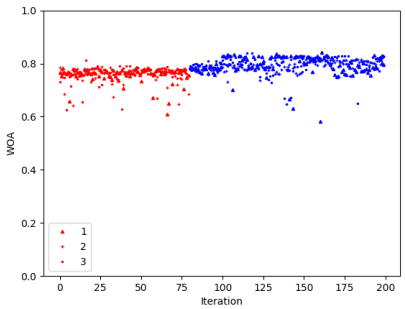
Random Forest



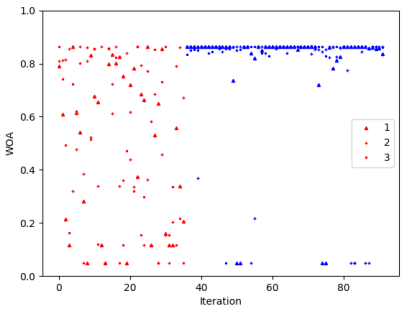
Extremely Randomized Trees



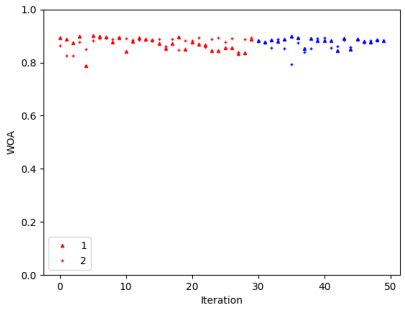
Stochastic Gradient Boosting



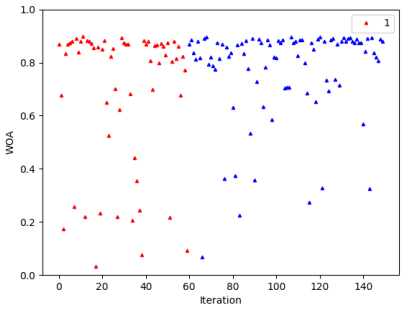
Extreme Gradient Boosting



Support Vector Machine



CNN architecture search



CNN training hyperparameters

Figure 20: Convergence over multiple BO optimization over the course of its iterations for all classifiers on the LCVF data. The initial Latin hypercube samples are shown in red.

As the CNN contains conditional variables dependant on the number of layers, it is the only model optimized using the TPE based hyperopt. The way this is implemented is by a 5-way choice for the number of layers, followed by optimizing the other parameters such as the filter size in each layer. This considers parameters in the same layer in models of different sizes as different hyperparameters, e.g. the filter size in the second layer of a network with 2 layers will be considered different from the filter size in the second layer of a 5 layer network. It would be interesting to see if performance can be improved by adding a binary choice at each layer, whether to add another layer or not, instead of choosing the number of layers at the start. This approach would consider hyperparameters in the same layer as the same hyperparameter, irrespective of the depth of the network.

Even though BO can be parallelized, for example using GPyOpt with local penalization, it will still be slower than grid and random search, because the whole batch has to be finished before the surrogate function can be updated. In other words, the slowest model in the batch decides how quickly the batch is completed and the configurations for the next batch can be determined. Furthermore, the calculation of the surrogate function itself can be costly, with GPs for instance having a time complexity cubic in the number of hyperparameters. Note that there do exist some more involved GP alternatives that reduce the computational complexity [69].

In order to determine if the optimization procedures carried out here as a whole resulted in finding better configurations for common models, we take a look at other scores reported in the literature on the same datasets. As our LCVF version is not publicly available, and IP does not come with a designated training and testing split, we focus on the UH data to compare weighted overall accuracies and kappa scores. In Table 18 we compare our results with those of Ghamisi et al. [30] and Mou, Ghamisi, and Zhu [53].

This table shows the usefulness of the kappa statistic. Purely going by WOA it appears that Mou, Ghamisi, and Zhu [53] found an excellent architecture; a model with a single convolutional layer scoring 5 percentage points higher than anything reported in this work. The kappa statistic however reveals that this is not the case. This raises the question if the hyperparameter optimization algorithms would find better models if they use the kappa statistic to optimize for instead of the WOA. No discrepancies between WOA and kappa score as large as those in Mou, Ghamisi, and Zhu [53] are present in our findings.

Even though the train and test samples used are exactly the same, the small differences in performance given in the table can still in part be attributed to other factors than just the hyperparameter configurations. Implementation details will have an impact, such as the scikit-learn RF implementation averaging the probabilities instead of letting each classifier vote for a single class, or using the one-versus-one instead of one-versus-many approach for the SVM. Additionally, details not specified such as the batch size or optimizer used for the CNN will also impact the final scores. Furthermore, the inherent randomness of classifiers such as the order in which training samples will be presented to a neural network can also have an impact.

Table 18: Performance of overlapping classifiers between studies on the UH dataset.

	Ghamisi [30]		Mou [53]		Doorenbos	
	Acc	κ	Acc	κ	Acc	κ
RF	72.99	70.97	72.93	70.91	73.13	71.26
SVM	80.18	78.66	77.09	75.36	79.17	77.58
CNN	78.21	78.46	85.42	72.00	79.28	77.56

All in all, from both Table 18 and Figure 20, there does not seem to be an indication that BO is able to find better models than those already reported in the literature.

8 Conclusion

We were able to provide a unique insight into the comparative performance of several modern classifiers on remotely sensed hyperspectral imagery, by actively addressing the important matter of hyperparameter optimization.

Out of the seven models considered in this context, the results presented in this work show that the SVM is an excellent choice. It outperforms the five tree ensembles, and reaches a higher score than the CNN on 2 out of the 3 images, while being very fast to train. Furthermore, the combination of its speed and only having 2 hyperparameters make it easy to optimize, with the BO loops used in our comparison generally finishing well before the full training of one single CNN model.

Out of the tree ensembles, the ERT lends itself as a good baseline that does not require much tuning, slightly outperforming RF in all aspects. There does not seem to be a use case where either bagging or SGB is the optimal choice of classifier, due to their relatively low accuracies and high training times. While XGB achieves decent performance, the tuning procedure is unintuitive and the training times are relatively high.

The 1-dimensional CNN achieves high scores, but due to the considerable amount of time needed for tuning and training, as well as the difficulty of its optimization, it should not be used as the first choice of classifier. Nevertheless, it can produce good results given enough time.

The classifiers considered did not seem to benefit from using a more involved hyperparameter optimization algorithm than grid or random search. Only in two cases, when directly optimizing for test WOA, did BO manage to find a configuration scoring over 1 standard deviation higher than the other strategies.

In further research the results in this work can be used for a comparison with the performance of other underrepresented models such as Deep Forest, or as a baseline to investigate the effects of dimensionality reduction [82]. Reducing the dimensionality can for instance be achieved by techniques such as Principal

Component Analysis, or through the manual selection of bands by a domain expert.
Alternatively, while we did not find large differences in model performance using different optimization strategies, these could occur when running them for a lower or higher number of iterations. Thus, additional research can be done comparing the effectiveness of these strategies as a function of the number of iterations.

9 Acknowledgements

I would like to thank Dr. Pedram Ghamisi for providing the University of Houston dataset and Dr. Erzsébet Merényi for providing the Lunar Crater Volcanic Field dataset.

References

[1] *2013 IEEE GRSS Data Fusion Contest*. `http://www.grss-ieee.org/community/technical-committees/data-fusion/`.

[2] Jacopo Acquarelli et al. "Spectral-spatial classification of hyperspectral images: Three tricks and a new learning setting". In: *Remote Sensing* 10.7 (2018), p. 1156.

[3] The GPyOpt authors. *GPyOpt: A Bayesian Optimization framework in Python*. `http://github.com/SheffieldML/GPyOpt`. 2016.

[4] James S Bergstra et al. "Algorithms for hyper-parameter optimization". In: *Advances in neural information processing systems*. 2011, pp. 2546–2554.

[5] James Bergstra and Yoshua Bengio. "Random search for hyper-parameter optimization". In: *Journal of machine learning research* 13.Feb (2012), pp. 281–305.

[6] James Bergstra, Daniel Yamins, and David Daniel Cox. "Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures". In: *Proceedings of Machine Learning Research* (2013).

[7] Rohit Uttam Bhagwat and B Uma Shankar. "A novel multilabel classification of remote sensing images using XGBoost". In: *2019 IEEE 5th International Conference for Convergence in Technology (I2CT)*. IEEE. 2019, pp. 1–5.

[8] Christopher M Bishop. *Pattern recognition and machine learning*. Springer, 2006.

[9] Leo Breiman. "Bagging predictors". In: *Machine learning* 24.2 (1996), pp. 123–140.

[10] Leo Breiman. "Random forests". In: *Machine learning* 45.1 (2001), pp. 5–32.

[11] Leo Breiman et al. *Classification and regression trees*. CRC press, 1984.

[12] Eric Brochu, Vlad M Cora, and Nando De Freitas. "A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning". In: *arXiv preprint arXiv:1012.2599* (2010).

[13] Jonathan Cheung-Wai Chan et al. "An evaluation of ensemble classifiers for mapping Natura 2000 heathland in Belgium using spaceborne angular hyperspectral (CHRIS/Proba) imagery". In: *International Journal of Applied Earth Observation and Geoinformation* 18 (2012), pp. 13–22.

[14] Tianqi Chen and Carlos Guestrin. "Xgboost: A scalable tree boosting system". In: *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. 2016, pp. 785–794.

[15] Yushi Chen et al. "Automatic design of convolutional neural network for hyperspectral image classification". In: *IEEE Transactions on Geoscience and Remote Sensing* 57.9 (2019), pp. 7048–7066.

[16] François Chollet. *Keras*. `https://github.com/fchollet/keras`. 2015.

[17] Jacob Cohen. "A coefficient of agreement for nominal scales". In: *Educational and psychological measurement* 20.1 (1960), pp. 37–46.

[18] Corinna Cortes and Vladimir Vapnik. "Support-vector networks". In: *Machine learning* 20.3 (1995), pp. 273–297.

[19] Michele Dalponte et al. "Tree species classification in boreal forests with hyperspectral data". In: *IEEE Transactions on Geoscience and Remote Sensing* 51.5 (2012), pp. 2632–2645.

[20] Erik Daxberger et al. "Mixed-Variable Bayesian Optimization". In: *arXiv preprint arXiv:1907.01329* (2019).

[21] Binu Melit Devassy and Sony George. "Ink Classification Using Convolutional Neural Network". In: *NISK Journal* 12 (2019).

[22] Ian Dewancker, Michael McCourt, and Scott Clark. *Bayesian optimization primer*. 2015.

[23] Richard O Duda, Peter E Hart, and David G Stork. *Pattern classification*. John Wiley & Sons, 2012.

[24] Matthias Feurer, Jost Tobias Springenberg, and Frank Hutter. "Initializing bayesian hyperparameter optimization via meta-learning". In: *Twenty-Ninth AAAI Conference on Artificial Intelligence*. 2015.

[25] Peter I Frazier. "A tutorial on bayesian optimization". In: *arXiv preprint arXiv:1807.02811* (2018).

[26] Jerome H Friedman. "Greedy function approximation: a gradient boosting machine". In: *Annals of statistics* (2001), pp. 1189–1232.

[27] Jerome H Friedman. "Stochastic gradient boosting". In: *Computational statistics & data analysis* 38.4 (2002), pp. 367–378.

[28] Eduardo C Garrido-Merchán and Daniel Hernández-Lobato. "Dealing with categorical and integer-valued variables in bayesian optimization with gaussian processes". In: *Neurocomputing* 380 (2020), pp. 20–35.

[29] Pierre Geurts, Damien Ernst, and Louis Wehenkel. "Extremely randomized trees". In: *Machine learning* 63.1 (2006), pp. 3–42.

[30] Pedram Ghamisi et al. "Advanced spectral classifiers for hyperspectral images: A review". In: *IEEE Geoscience and Remote Sensing Magazine* 5.1 (2017), pp. 8–32.

[31] Javier González et al. "Batch bayesian optimization via local penalization". In: *Artificial intelligence and statistics*. 2016, pp. 648–657.

[32] Lukas Hahn et al. "Fast and Reliable Architecture Selection for Convolutional Neural Networks". In: *arXiv preprint arXiv:1905.01924* (2019).

[33] Jisoo Ham et al. "Investigation of the random forest framework for classification of hyperspectral data". In: *IEEE Transactions on Geoscience and Remote Sensing* 43.3 (2005), pp. 492–501.

[34] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning: data mining, inference, and prediction*. Springer Science & Business Media, 2009.

[35] Matthew W Hoffman and Bobak Shahriari. "Modular mechanisms for Bayesian optimization". In: *NIPS workshop on Bayesian optimization*. Citeseer. 2014, pp. 1–5.

[36] Chih-Wei Hsu, Chih-Chung Chang, Chih-Jen Lin, et al. *A practical guide to support vector classification*. 2003.

[37] Wei Hu et al. "Deep convolutional neural networks for hyperspectral image classification". In: *Journal of Sensors* 2015 (2015).

[38] F. Hutter, H. Hoos, and K. Leyton-Brown. "An Efficient Approach for Assessing Hyperparameter Importance". In: *Proceedings of International Conference on Machine Learning 2014 (ICML 2014)*. June 2014, pp. 754–762.

[39] Shulong Jiang et al. "A novel framework for remote sensing image scene classification". In: *International Archives of the Photogrammetry, Remote Sensing & Spatial Information Sciences* 42.3 (2018).

[40] Donald R Jones, Matthias Schonlau, and William J Welch. "Efficient global optimization of expensive black-box functions". In: *Journal of Global optimization* 13.4 (1998), pp. 455–492.

[41] Nitish Shirish Keskar et al. "On large-batch training for deep learning: Generalization gap and sharp minima". In: *arXiv preprint arXiv:1609.04836* (2016).

[42] Muhammad Jaleed Khan et al. "Modern trends in hyperspectral image analysis: a review". In: *IEEE Access* 6 (2018), pp. 14118–14129.

[43] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[44] Sarawak Kuching. "The performance of maximum likelihood, spectral angle mapper, neural network and decision tree classifiers in hyperspectral image analysis". In: *Journal of Computer Science* 3.6 (2007), pp. 419–423.

[45] Chandan Kumar et al. "Automated lithological mapping by integrating spectral enhancement techniques and machine learning algorithms using AVIRIS-NG hyperspectral data in Gold-bearing granite-greenstone rocks in Hutti, India". In: *International Journal of Applied Earth Observation and Geoinformation* 86 (2020), p. 102006.

[46] Rick Lawrence et al. "Classification of remotely sensed imagery using stochastic gradient boosting as a refinement of classification tree analysis". In: *Remote sensing of environment* 90.3 (2004), pp. 331–336.

[47] Ying Li, Haokui Zhang, and Qiang Shen. "Spectral–spatial classification of hyperspectral imagery with 3D convolutional neural network". In: *Remote Sensing* 9.1 (2017), p. 67.

[48] Kyle Loggenberg et al. "Modelling water stress in a shiraz vineyard using hyperspectral imaging and machine learning". In: *Remote Sensing* 10.2 (2018), p. 202.

[49] David A. Landgrebe Marion F. Baumgardner Larry L. Biehl. *220 Band AVIRIS Hyperspectral Image Data Set: June 12, 1992 Indian Pine Test Site 3*. Sept. 2015. DOI: `doi:/10.4231/R7RX991C`. URL: `https://purr.purdue.edu/publications/1947/1`.

[50] Michael D McKay, Richard J Beckman, and William J Conover. "A comparison of three methods for selecting values of input variables in the analysis of output from a computer code". In: *Technometrics* 42.1 (2000), pp. 55–61.

[51] Michael J Mendenhall and Erzsébet Merényi. "Relevance-based feature extraction for hyperspectral images". In: *IEEE Transactions on Neural Networks* 19.4 (2008), pp. 658–672.

[52] Erzsébet Merényi et al. "Classification of hyperspectral imagery with neural networks: comparison to conventional tools". In: *EURASIP Journal on Advances in Signal Processing* 2014.1 (2014), p. 71.

[53] Lichao Mou, Pedram Ghamisi, and Xiao Xiang Zhu. "Deep recurrent neural networks for hyperspectral image classification". In: *IEEE Transactions on Geoscience and Remote Sensing* 55.7 (2017), pp. 3639–3655.

[54] G Narendra and D Sivakumar. "Deep Learning Based Hyperspectral Image Analysis—A Survey". In: *Journal of Computational and Theoretical Nanoscience* 16.4 (2019), pp. 1528–1535.

[55] Ary Noviyanto and Waleed H Abdulla. "Honey botanical origin classification using hyperspectral imaging and machine learning". In: *Journal of Food Engineering* 265 (2020), p. 109684.

[56] Chigozie Nwankpa et al. "Activation functions: Comparison of trends in practice and research for deep learning". In: *arXiv preprint arXiv:1811.03378* (2018).

[57] Lucas Prado Osco et al. "A Machine Learning Framework to Predict Nutrient Content in Valencia-Orange Leaf Hyperspectral Measurements". In: *Remote Sensing* 12.6 (2020), p. 906.

[58] Thais Mayumi Oshiro, Pedro Santoro Perez, and José Augusto Baranauskas. "How many trees in a random forest?" In: *International workshop on machine learning and data mining in pattern recognition*. Springer. 2012, pp. 154–168.

[59] Mahesh Pal and PM Mather. "Support vector machines for classification in remote sensing". In: *International journal of remote sensing* 26.5 (2005), pp. 1007–1011.

[60] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[61] Antonio Plaza et al. "Recent advances in techniques for hyperspectral image processing". In: *Remote sensing of environment* 113 (2009), S110–S122.

[62] Philipp Probst and Anne-Laure Boulesteix. "To tune or not to tune the number of trees in random forest". In: *The Journal of Machine Learning Research* 18.1 (2017), pp. 6673–6690.

[63] Sayan Putatunda and Kiran Rama. "A comparative analysis of hyperopt as against other approaches for hyper-parameter optimization of XGBoost". In: *Proceedings of the 2018 International Conference on Signal Processing and Machine Learning*. 2018, pp. 6–10.

[64] Laura Elena Raileanu and Kilian Stoffel. "Theoretical comparison between the gini index and information gain criteria". In: *Annals of Mathematics and Artificial Intelligence* 41.1 (2004), pp. 77–93.

[65] Sebastian Raschka. *Python Machine Learning*. Birmingham, UK: Packt Publishing, 2015. ISBN: 1783555130.

[66] Sebastian Ruder. "An overview of gradient descent optimization algorithms". In: *arXiv preprint arXiv:1609.04747* (2016).

[67] Alim Samat et al. "Ensemble Extreme Learning Machines for Hyperspectral Image Classification". In: *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* 7.4 (2014), pp. 1060–1069.

[68] Andrew M Saxe et al. "On random weights and unsupervised feature learning." In: *ICML*. Vol. 2. 3. 2011, p. 6.

[69] Bobak Shahriari et al. "Taking the human out of the loop: A review of Bayesian optimization". In: *Proceedings of the IEEE* 104.1 (2015), pp. 148–175.

[70] Rei Sonobe et al. "Crop classification from Sentinel-2-derived vegetation indices using ensemble learning". In: *Journal of Applied Remote Sensing* 12.2 (2018), p. 026019.

[71] Rei Sonobe et al. "Monitoring Photosynthetic Pigments of Shade-Grown Tea from Hyperspectral Reflectance". In: *Canadian Journal of Remote Sensing* 44.2 (2018), pp. 104–112.

[72] Jinya Su et al. "Aerial Visual Perception in Smart Farming: Field Study of Wheat Yellow Rust Monitoring". In: *IEEE Transactions on Industrial Informatics* (2020).

[73] Chris Thornton et al. "Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms". In: *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2013, pp. 847–855.

[74] Yan Wang and Xuelei Sherry Ni. "A XGBoost risk model via feature selection and Bayesian hyper-parameter optimization". In: *arXiv preprint arXiv:1901.08433* (2019).

[75] Björn Waske et al. "Mapping of hyperspectral AVIRIS data using machine-learning algorithms". In: *Canadian Journal of Remote Sensing* 35.sup1 (2009), S106–S116.

[76] Jeanette Weaver et al. "A Comparison of Machine Learning Techniques to Extract Human Settlements from High Resolution Imagery". In: *IGARSS 2018-2018 IEEE International Geoscience and Remote Sensing Symposium*. IEEE. 2018, pp. 6412–6415.

[77] Hao Wu and Saurabh Prasad. "Convolutional recurrent neural networks for hyperspectral data classification". In: *Remote Sensing* 9.3 (2017), p. 298.

[78] Junshi Xia et al. "Hyperspectral remote sensing image classification based on rotation forest". In: *IEEE Geoscience and Remote Sensing Letters* 11.1 (2013), pp. 239–243.

[79] Yufei Xia et al. "A boosted decision tree approach using Bayesian hyper-parameter optimization for credit scoring". In: *Expert Systems with Applications* 78 (2017), pp. 225–241.

[80] Xiliang Zhang et al. "Novel soft computing model for predicting blast-induced ground vibration in open-pit mines based on particle swarm optimization and XGBoost". In: *Natural Resources Research* (2019), pp. 1–11.

[81] Liheng Zhong, Lina Hu, and Hang Zhou. "Deep learning based multi-temporal crop classification". In: *Remote sensing of environment* 221 (2019), pp. 430–443.

[82] Zhi-Hua Zhou and Ji Feng. "Deep forest". In: *arXiv preprint arXiv:1702.08835* (2017).